PROGRAMMARE L'AMIGA

Eugene P. Mortimore

VOLUME II

IHT
GRUPPO EDITORIALE

IHT GRUPPO EDITORIALE
DIVISIONE LIBRI

COLLANA INFORMATICA

VOLUME II

PROGRAMMARE L'AMIGA

Programmare L'AMIGA

VOLUME II/PRIMA EDIZIONE

EUGENE P. MORTIMORE

Una pubblicazione
IHT Gruppo Editoriale S.r.l.
Via Monte Napoleone, 9
20121 Milano

Titolo originale dell'opera:
Amiga Programmer's Handbook, Volume II
Edizione in lingua inglese:

SYBEX Inc., Alameda, CA, USA

Direzione editoriale della collana di Massimiliano Lisa
Traduzione di Emanuele Cipolloni
Revisione di Luca Giachino
Collaborazione alla revisione di Mauro Gaffo
Impaginazione a cura di Antonio Gaviraghi e Andrea De Michelis

Disegno di copertina a cura di Thomas Ingalls & Associates
Grafica dell'interno a cura di Amparo Del Rio

ISBN 88-7803-005-8

Prima edizione: settembre 1989

Dedico questo libro
a una meravigliosa signora
chiamata Mary Alice.

RINGRAZIAMENTI

Desidero ringraziare Neil Katin, creatore dei dispositivi Timer e TrackDisk dell'Amiga, per l'aiuto recatomi nella soluzione di questioni tecniche. Anche Robert Peck, autore del libro *Programmer's Guide to the Amiga* pubblicato dalla SYBEX ha contribuito alla stesura dell'opera con importanti informazioni sui dispositivi.

Alla Commodore Business Machines Inc., dipartimento di supporto tecnico per l'Amiga, ringrazio Lisa A., Siracusa e Philip Lindsey per avermi aiutato nella ricerca d'informazioni aggiornate e nell'affrontare diversi problemi tecnici.

Desidero anche ringraziare Rudolph Langer e Karl Ray per i loro utili suggerimenti e per la loro guida nell'organizzazione di questo progetto.

Voglio ringraziare in special modo Valerie Robbins, redattrice della SYBEX, per la sua guida nell'organizzare, preparare e correggere le bozze di questo libro. È sempre un piacere lavorare con te, Valerie!

Kay Luthin merita un ringraziamento speciale per la sua attenta e paziente correzione delle bozze e il suo contributo all'organizzazione del libro. Grazie Kay, sei stata di grande aiuto!

Anche a Olivia Shinomoto rivolgo un ringraziamento speciale per il suo esperto lavoro di stesura al word processor del manoscritto. Grazie Olivia!

Infine, ringrazio Gladys Varon, fotocompositrice, e Jeff Green, correttore di bozze.

CORREZIONE DEGLI ERRORI

Questo volume è un manuale tecnico, che per la sua complessità può ancora contenere degli errori nonostante l'accurato lavoro svolto per offrire un'elevata qualità di contenuti.

Nel caso i lettori identificassero degli errori, o avessero degli aggiornamenti da proporre, possono segnalare il tutto alla redazione che si occupa di questo volume. In questo modo nelle successive edizioni, il libro conterrà correzioni e aggiornamenti certamente preziosi per tutti i programmatori. Gli aggiornamenti o le correzioni di errori vanno segnalate in forma scritta in modo chiaro, completo e conciso in modo che l'errore e la sua correzione possano essere verificati rapidamente (se necessario si possono allegare anche dischi).

Spedire il materiale a:

IHT Gruppo Editoriale
Divisione Libri
Redazione Programmare l'Amiga Vol. II
Via Monte Napoleone, 9
20121 Milano

SOMMARIO

INTRODUZIONE

XXIV **Sommario degli argomenti**

XXVI **Cosa può fare l'Amiga**

XXVII **I dispositivi di I/O dell'Amiga**
Procedure di programmazione XXVII
Accesso condiviso ed esclusivo ai dispositivi XXX

XXXI **L'ambiente di programmazione dell'Amiga**
Il disco del Kickstart XXXI
Il disco sistema XXXII
Il disco del compilatore XXXIV

1 I DISPOSITIVI DI I/O

1 **Introduzione**

1 **Interazioni task-dispositivo**

2 **I dispositivi di I/O**
Le richieste di I/O: messaggi per i dispositivi, 2
Il flusso delle richieste di I/O, 4

6 **Classi delle richieste di I/O**
I/O accodato, 6
I/O veloce, 8

9 **Interazioni con più di una reply port e con più di un'unità**
Un task con più reply port, 9
Interazioni di un task con più unità, 10

11 **Comportamento delle code**
Comportamento della coda alla request port, 11
Comportamento della coda alla reply port, 12

13 **Accesso condiviso ed esclusivo**

15 **Gestione multitasking ed elaborazione delle richieste di I/O**

17 **I dispositivi dell'Amiga**

19 **I comandi standard dei dispositivi**

20 **Le funzioni dei dispositivi**

22 **Le strutture che intervengono nei rapporti fra task e dispositivi**

24 **Strutture di I/O generali nel sistema Amiga**
La struttura IORequest, 24
La struttura IOStdReq, 26
La struttura Unit, 27
La struttura MsgPort, 28
La struttura Message, 30

31 **I file INCLUDE e le strutture relative ai dispositivi**

2 LA GESTIONE DEI DISPOSITIVI

37 **Introduzione**

37 **Procedure di programmazione generali**
Elaborazione delle richieste di I/O asincrono, 41
Elaborazione delle richieste di I/O sincrono, 45
Interazioni multiple fra task, reply port e dispositivi, 47
Elaborazione delle richieste di I/O in modo immediato, 49

51 **Procedure generali di gestione delle richieste di I/O**
Classi delle richieste di I/O, 53
Creazione di richieste multiple, 54
Elaborazione di più richieste di I/O, 55

57 **Le funzioni di accesso ai dispositivi**
La funzione AbortIO, 57
La funzione BeginIO, 58
La funzione sincrona DoIO, 60
La funzione asincrona SendIO, 61
La funzione sincrona WaitIO, 61
La funzione asincrona CheckIO, 62

63 **Le funzioni AddDevice e RemDevice**
AddDevice, 63
RemDevice, 64

65 **Impiego delle funzioni di supporto alla libreria Exec**
CreateExtIO, 66
CreatePort, 67
CreateStdIO, 69
CreateTask, 70
DeleteExtIO, 71
DeletePort, 72
DeleteStdIO, 73
DeleteTask, 74
NewList, 75

3 IL DISPOSITIVO AUDIO

79 **Introduzione**

79 **Il sistema hardware del dispositivo Audio**

82 **L'array di combinazioni dei canali**

85 **La priorità d'assegnazione dei canali**
Protezione e sottrazione dei canali, 86

87 **La chiave d'assegnazione**
Come avvertire un task dell'imminente emissione di un suono, 90

90 **I comandi del dispositivo Audio**

93 **Le strutture del dispositivo Audio**
La struttura IOAudio, 94
La struttura AudChannel, 98

101 **Codici d'errore del dispositivo Audio**

103 **Impiego delle funzioni**
CloseDevice, 103
OpenDevice, 104

108 **Comandi standard del dispositivo**
CMD_CLEAR, 108
CMD_FLUSH, 109
CMD_READ, 111
CMD_RESET, 113
CMD_START, 114
CMD_STOP, 116
CMD_UPDATE, 118
CMD_WRITE, 119

123 **Comandi specifici del dispositivo**
ADCMD_ALLOCATE, 123
ADCMD_FINISH, 126
ADCMD_FREE, 128
ADCMD_LOCK, 129
ADCMD_PERVOL, 131
ADCMD_SETPREC, 133

ADCMD_WAITCYCLE, 134

4 IL DISPOSITIVO NARRATOR

139 **Introduzione**

139 **Elaborazione del testo**

140 **I comandi del dispositivo Narrator**
L'invio dei comandi
al dispositivo Narrator, 140

142 **Le strutture del dispositivo Narrator**
La struttura narrator_rb, 143
La struttura mouth_rb, 146

148 **Codici d'errore del dispositivo Narrator**

149 **Impiego delle funzioni**
CloseDevice, 149
CloseLibrary, 151
OpenDevice, 152
OpenLibrary, 155
Translate, 156

158 **Comandi standard del dispositivo**
CMD_FLUSH, 158
CMD_READ, 159
CMD_RESET, 161
CMD_START, 162
CMD_STOP, 163
CMD_WRITE, 163

5 IL DISPOSITIVO PARALLEL

171 **Introduzione**

171 **Le operazioni di lettura e scrittura**

174 **I comandi del dispositivo Parallel**
L'invio dei comandi
al dispositivo Parallel, 174
I parametri di default, 176

176 **Le strutture del dispositivo Parallel**
La struttura IOPArray, 176
La struttura IOExtPar, 177
I flag riconosciuti
dal dispositivo, 178

180 **Le condizioni di EOF**
Acquisizione di dati, 180
Trasmissione di dati, 181

181 **Impiego delle funzioni**
CloseDevice, 181
OpenDevice, 183

186 **Comandi standard del dispositivo**
CMD_FLUSH, 186
CMD_READ, 187
CMD_RESET, 189
CMD_START, 190
CMD_STOP, 191
CMD_WRITE, 192

193 **Comandi specifici del dispositivo**
PDCMD_QUERY, 193
PDCMD_SETPARAMS, 195

6 IL DISPOSITIVO SERIAL

199 **Introduzione**

199 **Le operazioni di lettura e scrittura**

202 **I comandi del dispositivo Serial**
L'invio dei comandi
al dispositivo Serial, 202

204 **Le strutture del dispositivo Serial**
La struttura IOTArray, 205
La struttura IOExtSer, 206
I flag che controllano
il comportamento
del dispositivo Serial, 208

212 **Le condizioni di EOF**
Acquisizione di dati, 213
Trasmissione di dati, 213

214 **Impiego delle funzioni**
CloseDevice, 214
OpenDevice, 215

218 **Comandi standard del dispositivo**
CMD_CLEAR, 218
CMD_FLUSH, 219
CMD_READ, 220
CMD_RESET, 223
CMD_START, 224
CMD_STOP, 225
CMD_WRITE, 226

228 **Comandi specifici del dispositivo**
SDCMD_BREAK, 228
SDCMD_QUERY, 230
SDCMD_SETPARAMS, 231

7 IL DISPOSITIVO INPUT

237 **Introduzione**

238 **Funzionamento del dispositivo Input**

239 **Impiego delle funzioni di gestione degli eventi di input**

242 **I comandi del dispositivo Input**
L'invio dei comandi al dispositivo Input, 242

243 **Le strutture del dispositivo Input**
La struttura InputEvent, 244

247 **Impiego delle funzioni**
CloseDevice, 247
OpenDevice, 249

250 **Comandi standard del dispositivo**
CMD_FLUSH, 250
CMD_RESET, 251
CMD_START, 252
CMD_STOP, 253

254 **Comandi specifici del dispositivo**
IND_ADDHANDLER, 254
IND_REMHANDLER, 255
IND_SETMPORT, 256
IND_SETMTRIG, 257
IND_SETMTYPE, 259
IND_SETPERIOD, 260
IND_SETTHRESH, 262
IND_WRITEEVENT, 263

8 IL DISPOSITIVO CONSOLE

267 **Introduzione**

267 **Funzionamento del dispositivo Console**
Le operazioni di lettura e scrittura del dispositivo Console, 269

272 **I comandi del dispositivo Console**

273 **Le strutture del dispositivo Console**
La struttura ConUnit, 275
La struttura KeyMap, 278
La struttura KeyMapNode, 280
La struttura KeyMapResource, 280

281 **Impiego delle funzioni**
CDInputHandler, 281
CloseDevice, 283
OpenDevice, 284
RawKeyConvert, 286

288 **Comandi standard del dispositivo**
CMD_CLEAR, 288
CMD_READ, 289
CMD_WRITE, 293

296 **Comandi specifici del dispositivo**
CD_ASKDEFAULTKEYMAP, 296
CD_ASKKEYMAP, 297

CD_SETDEFAULTKEYMAP, 298
CD_SETKEYMAP, 299

9 IL DISPOSITIVO KEYBOARD

303 **Introduzione**

304 **Funzionamento del dispositivo Keyboard**
Elaborazione degli eventi di input da tastiera, 305

305 **I comandi del dispositivo Keyboard**
L'invio dei comandi al dispositivo Keyboard, 306

308 **Le strutture del dispositivo Keyboard**

308 **Impiego delle funzioni**
CloseDevice, 308
OpenDevice, 309

311 **Comandi standard del dispositivo**
CMD_CLEAR, 311
CMD_RESET, 312

313 **Comandi specifici del dispositivo**
KBD_ADDRESETHANDLER, 313
KBD_READEVENT, 315
KBD_READMATRIX, 317
KBD_REMRESETHANDLER, 319
KBD_RESETHANDLERDONE, 320

10 IL DISPOSITIVO GAMEPORT

325 **Introduzione**

327 **Funzionamento del dispositivo Gameport**
Elaborazione degli eventi di input prodotti dal dispositivo Gameport, 329

331 **I comandi del dispositivo Gameport**
L'invio dei comandi al dispositivo Gameport, 331

332 **Le strutture del dispositivo Gameport**

332 **La struttura GamePortTrigger**

334 **Impiego delle funzioni**
CloseDevice, 334
OpenDevice, 335

337 **Comandi standard del dispositivo**
CMD_CLEAR, 337

338 **Comandi specifici del dispositivo**
GPD_ASKCTYPE, 338
GPD_ASKTRIGGER, 339
GPD_READEVENT, 340
GPD_SETCTYPE, 344
GPD_SETTRIGGER, 345

11 IL DISPOSITIVO PRINTER

349 **Introduzione**

351 **Funzionamento del dispositivo Printer**
L'invio dei codici di controllo a una stampante, 353

355 **I comandi del dispositivo Printer**
L'invio dei comandi al dispositivo Printer, 355

358 **L'unione PrinterIO**

359 **Le strutture del dispositivo Printer**
La struttura IOPrtCmdReq, 360
La struttura IODRPReq, 362

363 **Impiego delle funzioni**
CloseDevice, 363
OpendDevice, 364

PWrite, 367

369 **Comandi standard del dispositivo**
CMD_FLUSH, 369
CMD_RESET, 370
CMD_START, 370
CMD_STOP, 371
CMD_WRITE, 372

374 **Comandi specifici del dispositivo**
PRD_DUMPRPORT, 374
PRD_PRTCOMMAND, 377
PRD_QUERY, 379
PRD_RAWWRITE, 380

12 IL DISPOSITIVO CLIPBOARD

385 **Introduzione**

385 **Il dispositivo Clipboard**
I clip privati, 385
I clip pubblici, 386

387 **Funzionamento del dispositivo Clipboard**
Gli identificatori dei clip, 388
Operazioni sequenziali di lettura e scrittura, 389

392 **I comandi del dispositivo Clipboard**
L'invio dei comandi al dispositivo Clipboard, 392

393 **Le strutture del dispositivo Clipboard**
La struttura ClipboardUnitPartial, 394
La struttura IOClipReq, 394
La struttura SatisfyMsg, 396

397 **Impiego delle funzioni**
CloseDevice, 397
OpenDevice, 398

400 **Comandi standard del dispositivo**
CMD_READ, 400
CMD_RESET, 402
CMD_UPDATE, 403
CMD_WRITE, 404

407 **Comandi specifici del dispositivo**
CBD_CURRENTREADID, 407
CBD_CURRENTWRITEID, 408
CBD_POST, 410

13 IL DISPOSITIVO TIMER

415 **Introduzione**

415 **Funzionamento del dispositivo Timer**

417 **Le unità del dispositivo Timer**
Correzione dei tempi in un sistema fortemente occupato, 418

421 **I comandi del dispositivo Timer**
L'invio dei comandi al dispositivo Timer, 421

422 **Le strutture del dispositivo Timer**
La struttura timeval, 423
La struttura timerequest, 423

424 **Impiego delle funzioni**
AddTime, 424
CloseDevice, 426
CmpTime, 427
OpenDevice, 428
SubTime, 430

431 **Comandi specifici del dispositivo**
TR_ADDREQUEST, 431
TR_GETSYSTIME, 432
TR_SETSYSTIME, 434

14 IL DISPOSITIVO TRACKDISK

437 **Introduzione**

437 **Funzionamento del dispositivo TrackDisk**
Interazioni tra disco e dispositivo TrackDisk, 439

441 **I comandi del dispositivo TrackDisk**
L'invio dei comandi al dispositivo TrackDisk, 441

442 **Le strutture del dispositivo TrackDisk**
La struttura IOExtTD, 443
Il parametro iotd_Count e i comandi estesi, 445
La struttura TDU_PublicUnit, 445
La struttura BootBlock, 447

447 **Codici d'errore del dispositivo TrackDisk**

449 **Impiego delle funzioni**
CloseDevice, 449
OpenDevice, 450

451 **Comandi specifici del dispositivo**
ETD_CLEAR, CMD_CLEAR, 451
ETD_FORMAT, TD_FORMAT, 452
ETD_MOTOR, TD_MOTOR, 454
ETD_RAWREAD, TD_RAWREAD, 455
ETD_RAWWRITE, TD_RAWWRITE, 457
ETD_READ, CMD_READ, 458
ETD_SEEK, TD_SEEK, 460
ETD_UPDATE, CMD_UPDATE, 461
ETD_WRITE, CMD_WRITE, 462
TD_ADDCHANGEINT, 463
TD_CHANGENUM, 464
TD_CHANGESTATE, 465
TD_GETDRIVETYPE, 466
TD_GETNUMTRACKS, 467
TD_PROTSTATUS, 467
TD_REMCHANGEINT, 468
TD_REMOVE, 469

APPENDICE

473 **Definizioni in C delle funzioni di supporto alla libreria Exec**
CreateExtIO, 473
CreatePort, 474
CreateStdIO, 475
CreateTask, 476
DeleteExtIO, 477
DeletePort, 477
DeleteStdIO, 478
DeleteTask, 478

INDICE ANALITICO

Introduzione

Introduzione

Questo libro è il secondo di due volumi: nel primo si esaminano tutte le funzioni delle librerie Diskfont, Exec, Graphics, Icon, Intuition, Layers, mentre nel secondo si prendono in considerazione i 12 dispositivi software di I/O. I due volumi rappresentano nell'insieme una completa guida di riferimento per il programmatore che desidera usare le librerie e i dispositivi dell'Amiga: per questa ragione, si raccomanda al lettore anche la consultazione del volume I.

Le notizie riportate in questo manuale si possono applicare a tutte e tre le macchine della serie Amiga esistenti oggi: il capostipite Amiga 1000 e i successivi Amiga 500 e 2000. Questi computer sono tutti compatibili tra loro per quanto riguarda il software, dal momento che adottano lo stesso sistema operativo. Qualsiasi comando o funzione della versione 1.3 documentato in questo libro lavora allo stesso modo su ciascuna delle tre macchine; un programma scritto in modo appropriato per un computer verrà eseguito correttamente anche sugli altri. L'unica differenza consiste nel diverso modo di trattare le librerie di sistema: mentre l'Amiga 1000 utilizza il disco del Kickstart per caricare in memoria la maggior parte di queste librerie, l'Amiga 500 e l'Amiga 2000 contengono il software sistema direttamente in ROM.

I dispositivi dell'Amiga sono preprogrammati in modo insolitamente efficiente, per un microcomputer. Le routine interne, che definiscono le funzioni e i comandi concernenti ciascun dispositivo, sono state verificate a fondo e hanno raggiunto una forma definitiva. Tuttavia, poiché queste routine sono definite a priori e quindi non sono modificabili, un programmatore deve conoscere appieno il loro funzionamento per riuscire a scrivere un programma che ne faccia un uso efficace. La comprensione dell'interazione tra i task e i dispositivi è d'importanza cruciale per la programmazione.

I file INCLUDE scritti in linguaggio C e in Assembly contengono tutte le informazioni necessarie al corretto funzionamento dei programmi. La difficoltà maggiore nella programmazione dei dispositivi risiede dunque nell'imparare le regole e nello scrivere programmi coerenti con queste regole. A questo scopo, bisogna comprendere a fondo i concetti che stanno alla base del funzionamento dei dispositivi nell'Amiga. Le figure presentate nei primi due capitoli di questo libro si propongono di illustrare le nozioni fondamentali della programmazione dei dispositivi, ovvero lo schema concettuale attraverso cui è possibile capire e programmare l'Amiga.

Quando si sono comprese a fondo le procedure relative alla programmazione dei dispositivi, occorre imparare il significato di tutte le strutture di dati che intervengono, dei loro parametri, dei parametri relativi ai dispositivi e ai flag. Solo così diventa possibile, usando con profitto le routine e le funzioni predefinite, scrivere i propri programmi di gestione dei dispositivi in maniera creativa e funzionale.

Sommario degli argomenti

Questo secondo volume è scritto come il primo in forma di manuale di riferimento. I primi due capitoli presentano alcuni concetti generali sulla programmazione dei dispositivi che vanno assimilati per poter procedere nel modo migliore. I 12 capitoli successivi prendono in esame ogni dispositivo singolarmente. Tutti i capitoli iniziano con un'analisi generale di ciò che è necessario sapere sul funzionamento del dispositivo, resa più immediata da alcune figure. Successivamente vengono esaminate in dettaglio le strutture, le funzioni e i comandi relativi al dispositivo.

Il primo capitolo descrive l'I/O con i dispositivi. compresa la modalità di accesso veloce (il QuickIO) e accodato (il QueuedIO), i segnali scambiati tra i task e i dispositivi sia singolarmente sia in concorrenza, il trattamento delle code e i modi di accesso condiviso ed esclusivo. Il capitolo elenca poi le strutture, i file INCLUDE, le funzioni e le librerie relative ai dispositivi: in questo contesto viene anche presentato il modello di message port che viene utilizzato per la maggior parte dei dispositivi. Questo modello mostra come venga usata una struttura Unit per svolgere ciascuna sessione di lavoro.

Il secondo capitolo esamina la gestione dei dispositivi, comprese le procedure necessarie alla programmazione in generale e alle strutture di richiesta di I/O (strutture di I/O). Si analizza così l'uso delle funzioni AbortIO, BeginIO, DoIO, SendIO, CheckIO e WaitIO, dei comandi comuni a tutti i dispositivi, nonché l'importanza delle funzioni di supporto alla libreria Exec: queste nove funzioni rendono molto semplice la gestione dei task, delle message port, delle strutture di I/O e delle code relative ai dispositivi.

Il terzo capitolo prende in esame il dispositivo Audio, che consente a un task di generare suoni tramite quattro canali audio. A causa del sistema multitasking e della presenza di più canali, questo dispositivo è il più complesso dell'intero sistema Amiga, sia per quanto riguarda la comprensione sia per la programmazione.

Il quarto capitolo riguarda il dispositivo Narrator, che permette a un task di far parlare l'Amiga per mezzo di fonemi e algoritmi che producono una voce sintetica. Fondamentali per questo dispositivo sono il dispositivo Audio e la funzione Translate della libreria Translator, che verranno opportunamente illustrate.

Il quinto capitolo esamina il dispositivo Parallel, preposto al controllo delle informazioni che vengono scambiate fra i task e le periferiche connesse alla porta parallela.

Il sesto capitolo riporta le analoghe informazioni relative al dispositivo Serial.

Nel settimo capitolo viene discusso il dispositivo Input, che serve per controllare il meccanismo degli eventi di input. Questo dispositivo agisce come un sistema di controllo misto delle informazioni provenienti dalla tastiera, dalle porte giochi, dal dispositivo Timer e dai disk drive. Un task può aggiungere eventi di input propri a questo flusso.

Nell'ottavo capitolo viene discusso il dispositivo Console, responsabile dell'elaborazione dei segnali provenienti dalla tastiera attraverso le routine del

dispositivo Keyboard, e dell'output formattato che viene inviato alle finestre di Intuition. Questo dispositivo interagisce con i dispositivi Input, TrackDisk, Keyboard, Timer e Gameport. Console non segue esattamente il modello presentato nel capitolo 1 riguardante le message port tra task e dispositivo, ma utilizza la struttura ConUnit, come sarà dettagliatamente spiegato.

Il nono capitolo prende in esame il dispositivo Keyboard, che permette a un task di controllare l'hardware di gestione della tastiera. Questo dispositivo viene utilizzato per far eseguire alla macchina particolari compiti in corrispondenza di opportune combinazioni di tasti definite dall'utente.

Nel decimo capitolo si prende in esame il dispositivo Gameport che consente a un task di controllare le porte giochi: questo dispositivo gestisce gli eventi di input generati da un mouse o da altri tipi di controller, come il joystick.

Nell'undicesimo capitolo viene analizzato il dispositivo Printer, che permette ai task di accedere alla stampante collegata alla porta seriale o a quella parallela. Il dispositivo Printer si occupa d'inviare testi e bitmap a una stampante, e si affida al tool Preferences per sapere a quale porta è collegata e quali sono le sue caratteristiche. In questo capitolo si discutono anche i driver di stampa.

Il dodicesimo capitolo esamina il dispositivo Clipboard, che rende possibili le operazioni di cut & paste. Questo dispositivo gestisce il trasferimento di dati tra i diversi task e tra i buffer dedicati al Clipboard all'interno di uno stesso task. Il dispositivo segue il modello presentato nel capitolo 1, ma spesso utilizza una message port addizionale.

Il tredicesimo capitolo analizza il dispositivo Timer, che permette a un task di controllare le temporizzazioni; questo dispositivo gestisce gli eventi temporizzati e i segnali che danno inizio all'attività di un task.

Il quattordicesimo capitolo, infine, analizza il dispositivo TrackDisk, che tramite i suoi comandi di bassissimo livello permette di effettuare operazioni inconsuete con i disk drive dell'Amiga.

L'appendice presenta le istruzioni che costituiscono le funzioni di supporto alla libreria Exec descritte nel capitolo 2. Sebbene queste funzioni siano presenti nelle librerie linked che intervengono in fase di compilazione, vengono ugualmente descritte nei dettagli per mostrare al programmatore come eseguire le procedure fondamentali per accedere ai dispositivi. Il sommario fornisce un'utile guida alle informazioni presentate in questo volume. Inoltre, per rintracciare rapidamente le strutture dei dispositivi, le funzioni e i comandi, ci si può riferire all'indice analitico.

Anche se questo libro non è studiato per insegnare come si possono creare dispositivi personalizzati, i diversi aspetti di un dispositivo sono illustrati da un gran numero di figure che possono essere d'aiuto nello studio di nuovi dispositivi da aggiungere a quelli dell'Amiga.

Cosa può fare l'Amiga

Il modo migliore per rendersi conto delle capacità dell'Amiga è vederlo al lavoro. Si immagini un nuovo centro commerciale che vuole usare un computer per offrire ai passanti una presentazione divertente per invogliarli a entrare. Usando un Amiga, la presentazione potrebbe essere realizzata in questo modo:

- offrendo un'attraente video di cinque minuti, con una voce fuori campo e una musica in sottofondo.
- Dando la possibilità di scegliere altre sequenze d'immagini, di descrizioni vocali o di musiche di sottofondo tramite alcune semplici combinazioni di tasti.
- Permettendo la scelta tra varie alternative per mezzo di un mouse che punta agli oggetti visualizzati sullo schermo.
- Richiedendo risposte (per una ricerca di mercato, ad esempio).
- Creando stampe laser a colori da offrire in omaggio.
- Consentendo alla direzione di controllare le risposte attraverso un modem allo scopo di determinare le preferenze dei clienti.
- Preparando per la direzione un disco contenente l'elenco completo delle interazioni tra l'Amiga e il cliente.

Con l'Amiga, tutto ciò è possibile: questo computer può produrre contemporaneamente musica stereofonica e voce sintetizzata di elevata qualità utilizzando dati preparati in precedenza, può rispondere alle richieste dell'utente, può stampare scritti e illustrazioni, e inviare informazioni a un modem attraverso la porta seriale. A questo scopo, l'Amiga mette a disposizione le seguenti caratteristiche:

- un'ampia memoria dove inserire i dati audio e video, che possono essere prodotti in anticipo e quindi memorizzati sul disco utilizzando algoritmi di compressione. La memoria richiesta normalmente per una presentazione audio e video di cinque minuti può raggiungere le dimensioni di un hard disk.
- La capacità di spostare velocemente le informazioni dalla fast RAM alla chip RAM, unica area di memoria alla quale possono accedere i chip custom dell'Amiga per elaborare dati come le bitmap e i suoni campionati.
- Un'interfaccia user-friendly, con ottime caratteristiche di velocità e semplicità.

- Un sistema operativo multitasking che permette ai task di scambiarsi informazioni e di segnalarsi l'un l'altro l'arrivo di queste informazioni.

- La capacità di accettare in ingresso dati forniti da svariate sorgenti esterne contemporaneamente (la tastiera, il mouse, le porte giochi, i disk drive e così via), di fonderle insieme in un unico flusso di dati e di tenerne conto senza interferire con la presentazione in corso.

- Un sistema veloce e funzionale, che può adattarsi simultaneamente a più categorie di dati.

- La capacità di adattarsi continuamente e senza intoppi a un ambiente funzionante in tempo reale, nel quale l'ordine degli eventi non è determinato a priori ma può cambiare a seconda delle scelte dell'utente.

Sono i dispositivi dell'Amiga che forniscono i mezzi per creare una presentazione come quella descritta e permettono dunque di sfruttare fino in fondo le capacità del computer.

dispositivi di I/O dell'Amiga

Sette dispositivi sono residenti in ROM, e sono Audio, Input, Console, Timer, Keyboard, Gameport e TrackDisk. Per l'Amiga 1000, le routine relative si trovano sul disco del Kickstart e vengono caricate in RAM la prima volta che si effettua il boot. Questa RAM è gestita in maniera particolare dal sistema, in quanto, una volta completato il caricamento viene impedito ogni ulteriore accesso in scrittura; questo particolare tipo di memoria viene chiamata ROM WCS (Write Control Store, ossia memoria a controllo di scrittura). A differenza dell'Amiga 1000, nei successivi modelli questi dispositivi sono stati inseriti direttamente in ROM. Gli altri cinque dispositivi, Narrator, Serial, Parallel, Printer e Clipboard sono invece residenti sul disco per tutte e tre le macchine, e vengono quindi caricati in memoria solo quando occorre.

Procedure di programmazione

In linea di massima, le procedure necessarie per accedere alle routine interne dei 12 dispositivi dell'Amiga sono sempre le stesse: un task apre un'unità del dispositivo tramite una chiamata alla routine OpenDevice dell'Exec, interagisce con essa e poi la chiude con una chiamata a CloseDevice. La prima volta che un dispositivo viene aperto, il sistema alloca automaticamente lo spazio necessario e inizializza una struttura di tipo Device per gestire il dispositivo, e una struttura di tipo Unit che rappresenta la message port dell'unità. Usando dispositivi con modalità di accesso condiviso, le stesse strutture Device e Unit possono essere condivise da tutti i task che hanno aperto quell'unità. I parametri lib_OpenCnt della struttura Device e unit_OpenCnt

della struttura Unit (che inizialmente hanno valore 0) vengono incrementati o decrementati di 1 ogni volta che un task apre o chiude l'unità.

Per interagire con un dispositivo, un task deve servirsi di un'apposita struttura di dati che chiamiamo genericamente struttura di I/O. In generale, si tratta di strutture Message (cioè messaggi) più o meno estese a seconda del numero di parametri che il dispositivo si aspetta di ricevere. Questi parametri identificano univocamente il dispositivo e l'unità con la quale si desidera comunicare, indicano al dispositivo il comando che deve eseguire, se è previsto uno scambio di dati gli segnalano dove si trovano i relativi buffer, e così via. Quando la struttura di I/O è pronta, il task non fa altro che inviarla al dispositivo, che ovviamente dev'essere già stato aperto. Come spiegheremo dettagliatamente, in realtà la struttura di I/O non si muove dall'area di memoria che occupa: è il suo indirizzo che viene inviato e utilizzato per accedere ai suoi parametri.

A questo punto, il dispositivo riceve la richiesta, rileva a quale delle sue unità è diretta e la accoda alla request port di quell'unità (la request port è la message port dell'unità nella cui "coda" vengono inserite le richieste a mano a mano che pervengono). La richiesta sale lungo la coda via via che l'unità soddisfa le richieste che la precedono. Quando arriva il suo turno, l'unità la estrae dalla coda, la elabora e successivamente la inoltra alla reply port del mittente, cioè il task che l'aveva inviata (la reply port è la message port del task nella cui coda vengono inseriti i messaggi a mano a mano che pervengono). Ora il task può accedervi e analizzare i parametri restituiti dall'unità. Perché questo tragitto si completi correttamente, occorre che il task che ha inviato la richiesta non vi acceda fino a quando l'unità non la restituisce.

Il tragitto descritto prevede accodamenti da entrambe le parti, prima alla request port dell'unità e poi alla reply port del task. Tuttavia, il sistema prevede alcuni meccanismi d'inoltro dei comandi che possono evitare l'accodamento all'inizio o alla fine del processo; se si ricorre a questi meccanismi, il task riceve i dati più in fretta di come accade con le code d'ingresso e di uscita.

Un task interagisce con le routine interne di un dispositivo inviando i comandi che specificano qual è il tipo di operazione richiesta. Questi comandi vengono interpretati dalle funzioni DoIO e SendIO della libreria Exec e dalla funzione BeginIO, che varia per ogni dispositivo (si veda il capitolo 2). In sostanza, un task può accedere a un dispositivo in lettura o in scrittura, e quasi tutte le operazioni relative ai dispositivi si riconducono alla lettura o alla scrittura di dati. La Figura I.1 (nella pagina successiva) descrive questo tipo di operazioni.

Se un task utilizza un buffer per definire i dati e poi inviarli a un dispositivo, si dice che il sistema sta eseguendo un'*operazione di scrittura*. In questo caso, i dati hanno origine nel buffer definito dal task entro il proprio spazio di memoria, passano poi attraverso la memoria del dispositivo e infine sono inviati all'hardware esterno. I dispositivi che possono scrivere dati (Audio, Serial, Parallel, Narrator, Printer, Clipboard e TrackDisk) possiedono almeno un comando di scrittura, che è indicato dalla parola "WRITE" compresa nel nome del comando.

Se un task richiede dati da un dispositivo hardware esterno e fa uso di un buffer da lui definito per riceverli, si dice invece che il sistema sta eseguendo

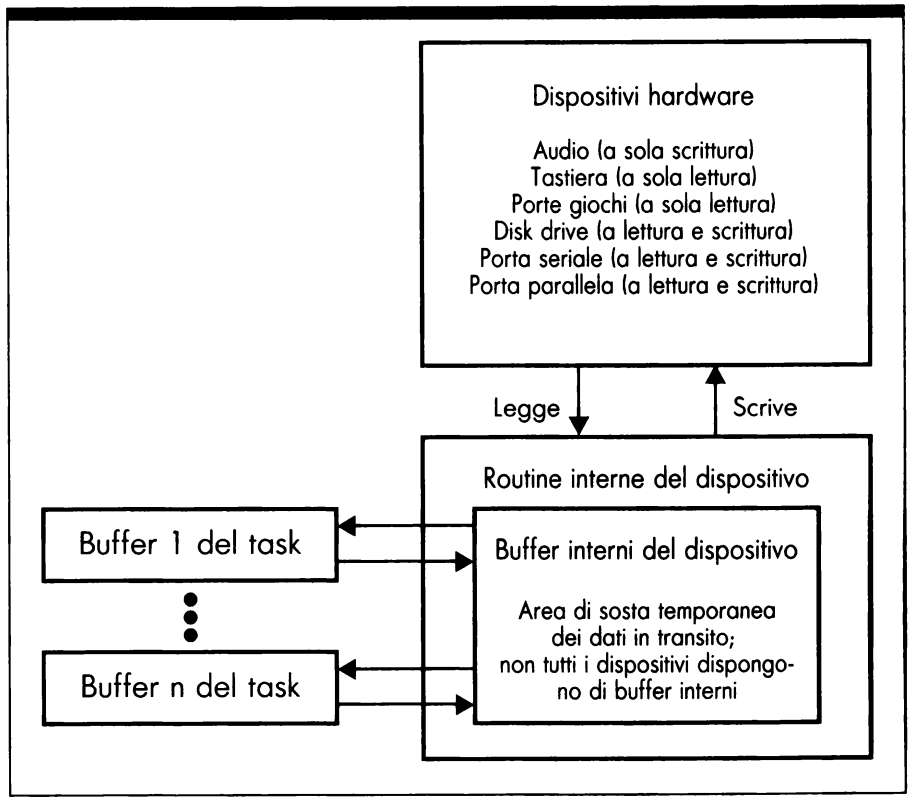


Figura I.1: *Operazioni di lettura e scrittura generali*

un'*operazione di lettura*. I dati di solito passano prima attraverso la memoria del dispositivo, per poi giungere nel buffer definito dal task in attesa di successivi accessi da parte di quest'ultimo. I dispositivi che possono leggere dati (Serial, Parallel, Narrator, Keyboard, Gameport, Clipboard e TrackDisk) hanno tutti a disposizione almeno un comando di lettura, individuato dalla parola "READ" nel nome del comando.

I dispositivi che consentono operazioni di lettura spesso fanno uso di un buffer "interno" che risiede in RAM ed è gestito automaticamente dalle loro routine interne. Si dice allora che il dispositivo "possiede" quello spazio di memoria, anche nel caso che le sue routine interne risiedano in ROM. Anche alcuni dispositivi che consentono operazioni in scrittura hanno a disposizione un buffer in RAM, gestito e preparato automaticamente dalle loro routine. Il programmatore si occupa soltanto dei buffer definiti dai task che crea. Sui buffer interni dei dispositivi i task possono solo compiere operazioni di cancellazione e di aggiornamento.

La maggior parte dei dispositivi restituisce un codice d'errore quando qualcosa non viene portato a termine correttamente durante l'elaborazione della richiesta di I/O. Questi errori sono restituiti in due modi: le funzioni OpenDevice, DoIO e WaitIO contenute nella libreria Exec restituiscono un

codice d'errore nelle variabili nelle quali vengono eguagliate (si veda il volume I). Inoltre, nella struttura di I/O restituita compaiono codici specifici e dettagliati per ogni dispositivo. Nel caso di OpenDevice il task deve solo controllare se la chiamata alla funzione restituisce un valore diverso da zero: se il valore non è zero, significa che si è verificato un errore durante l'esecuzione. Di conseguenza, il task dovrebbe lasciare libera tutta la memoria precedentemente occupata e avvertire l'utente oppure eseguire adeguate procedure di correzione.

È disponibile anche un livello più dettagliato di controllo degli errori. Qualsiasi errore verificatosi durante l'elaborazione da parte del dispositivo causa la restituzione di un valore diverso da zero nel parametro io_Error della struttura di I/O: per esempio, se una chiamata a OpenDevice fallisce, al parametro io_Error relativo alla sua struttura di I/O è assegnato il valore IOERR_OPENFAIL per indicare che non è stato possibile aprire il dispositivo. Nella maggior parte dei casi, il numero presente in io_Error fornisce al task tutte le informazioni necessarie a stabilire che cosa non ha funzionato. È quindi possibile confrontare il codice dell'errore con i valori presenti nei file INCLUDE per sapere qual è la ragione dell'insuccesso.

Accesso condiviso ed esclusivo ai dispositivi

Poiché l'Amiga è un sistema multitasking, spesso i task devono condividere le unità dei dispositivi. Ecco quali sono le linee di principio seguite nella progettazione e quindi da rispettare se si vuole aggiungere un dispositivo.

- Se un dispositivo invia dati verso un dispositivo hardware (o li riceve), di solito il sistema gli fornisce un meccanismo di collegamento esclusivo. I dispositivi Serial e Parallel lavorano entrambi con l'hardware, e per default permettono l'accesso esclusivo. Comunque, questi due dispositivi prevedono un flag che li fa diventare ad accesso condiviso. Poiché il dispositivo Printer lavora indirettamente sia con Serial sia con Parallel, inviando dati all'uno o all'altro a seconda di dov'è collegata la stampante, anch'esso dispone del modo di accesso esclusivo.

- Se un dispositivo non interagisce mai con l'hardware, tranne che per ricevere dati in input, le sue unità possono sempre essere condivise tra i task. I dispositivi Input, Console, Keyboard, Timer e Clipboard hanno questa caratteristica. Il dispositivo Narrator manda i suoi dati al dispositivo Audio, quindi lavora anche lui in accesso condiviso. Il dispositivo Audio, invece, sebbene mandi i suoi dati direttamente all'hardware, dispone di una complessa serie di regole che consentono a più task di accedere simultaneamente alle sue risorse.

L'ambiente di programmazione dell'Amiga

La Figura I.2 (nella pagina successiva) illustra l'ambiente di lavoro dell'Amiga. Mostra tre dischi, il disco del Kickstart dell'Amiga 1000 e i due dischi del sistema di programmazione in linguaggio C. L'organizzazione degli ultimi due dischi è analoga per tutti e tre i computer Amiga. In questo capitolo facciamo riferimento al compilatore C della Lattice, ma tutto ciò che diremo vale per ogni altro compilatore C (le uniche cose che cambiano sono al massimo i nomi dei tool di compilazione e di link).

Per sviluppare programmi in C sono richiesti alcuni file e alcuni dati che vanno inseriti nelle subdirectory riconosciute dal sistema. La sessione di lavoro inizia con il disco del Workbench nel disk drive interno e con il disco del compilatore nel disk drive aggiuntivo. È opportuno riorganizzare il contenuto di questi dischi allo scopo di risparmiare spazio: le sezioni successive danno alcuni suggerimenti su quello che è indispensabile conservare e su quello che si può tralasciare.

È anche opportuno, se si possiede abbastanza memoria RAM da poter creare un RAM disk tale da contenere la maggior parte dei file richiesti per la programmazione, creare un file comandi di startup che copi in esso i file di maggiore utilità.

La memoria RAM dell'Amiga consiste di almeno 512K di memoria interna e di un massimo di 8 MB di memoria aggiuntiva. I primi 512K di memoria possono essere usati come chip RAM (MEMF_CHIP, si veda il primo volume), mentre la memoria aggiuntiva viene configurata come fast RAM (MEMF_FAST).

Il disco del Kickstart

Per il boot, l'Amiga 1000 ha bisogno del disco del Kickstart, che contiene svariati settori relativi al boot e altre informazioni necessarie per inizializzare opportunamente la memoria e l'hardware. Il disco del Kickstart contiene anche la maggior parte delle routine interne relative ai dispositivi e alle librerie dell'Amiga che verranno usate nei programmi e durante la fase di debug effettuata tramite il ROMWack. Tutte queste informazioni vengono caricate nella ROM WCS (256K), che successivamente non è più alterabile.

Sull'Amiga 500 e 2000 queste informazioni sono inserite permanentemente nella memoria ROM. Per questi computer, quindi, la parte di sistema operativo contenente il Kickstart può essere aggiornata soltanto sostituendo il chip della ROM. Con l'Amiga 1000, quando il Kickstart è stato caricato in memoria, il disco va rimosso dal disk drive e al suo posto va inserito il disco sistema.

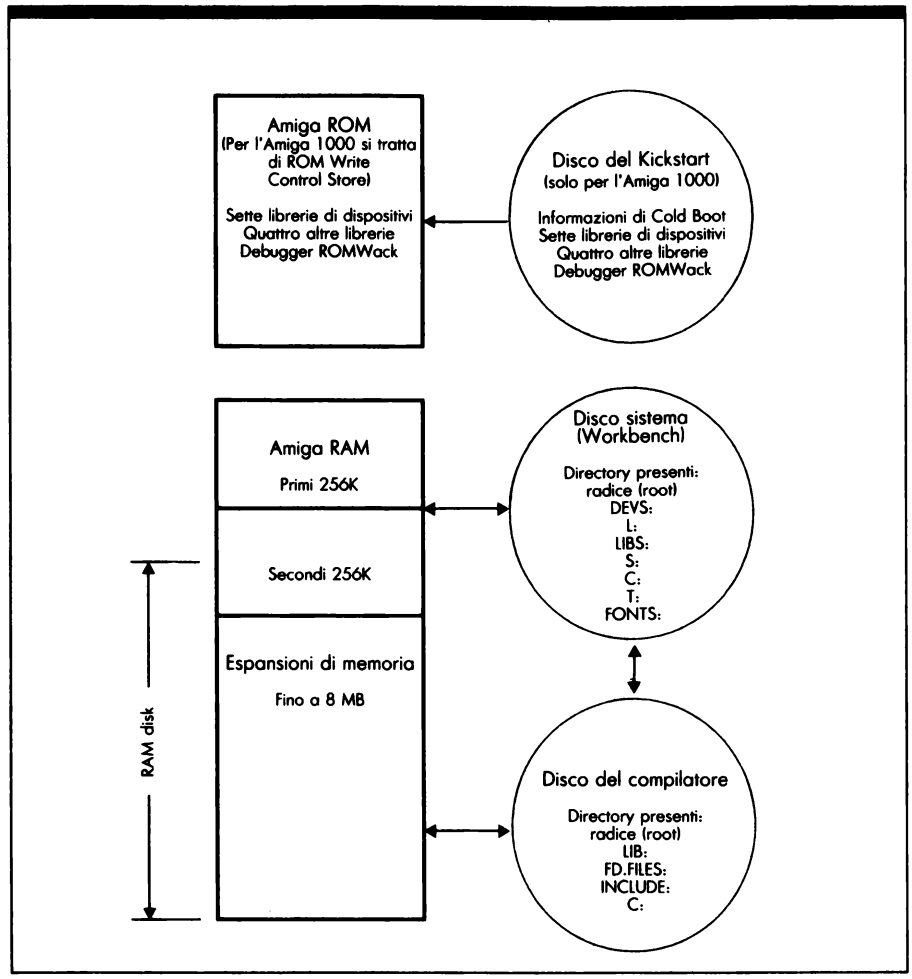


Figura I.2: *Ambiente di programmazione dell'Amiga*

Il disco sistema

Il disco da inserire nel disk drive interno è in pratica il disco del Workbench fornito dalla Commodore, da cui vengono eliminati alcuni file non necessari per la programmazione in C. Ecco l'elenco delle directory indispensabili.

- La directory radice. Contiene tutti i file sorgente per i quali c'è spazio sufficiente. È possibile lasciare alcuni file sorgente anche nella directory principale del secondo disco.

- La subdirectory devs. Contiene cinque librerie di dispositivi che non si trovano sul disco del Kickstart. Contiene anche il file Mountlist, il file di configurazione del sistema e una subdirectory contenente una serie

di driver per stampanti. È bene eliminare tutti i driver per stampanti non necessari. Il file Mountlist dovrebbe inoltre contenere i dati relativi a tutti i dispositivi hardware aggiuntivi perché il sistema possa trattarli come dispositivi logici.

- La subdirectory l. Contiene programmi necessari per il corretto funzionamento dell'Amiga, fra cui il Disk-Validator, che controlla i dischi quando vengono inseriti nei disk drive, il Ram-Handler, che gestisce il RAM disk nel quale vengono collocati i programmi da utilizzare per la programmazione e il Port-Handler, che gestisce la porta parallela e quella seriale.

- La subdirectory libs. Comprende diversi file di libreria contenenti funzioni di sistema che vengono rese disponibili solo all'occorrenza. Il file version.library gestisce il sistema e segnala qual è la versione di ciascuna libreria. Le librerie icon.library e info.library servono soltanto se si utilizzano le funzioni del Workbench (si veda il primo volume) per gestire le icone dello schermo. Queste librerie di funzioni vengono chiamate librerie shared.

- La subdirectory s. Contiene tutti i file comandi dell'AmigaDOS che vengono utilizzati durante la programmazione. In particolare, deve contenere il file comandi della startup-sequence, che definisce le operazioni di startup che hanno luogo quando l'Amiga viene inizializzato. Questo file può copiare nel RAM disk tutti i file necessari. La startup-sequence viene sempre eseguita quando si resetta o si accende la macchina.

- La subdirectory c. Contiene tutti i comandi del sistema operativo AmigaDOS, tra cui il comando DIR, il comando ADDBUFFERS, EXECUTE, RUN e molti altri comandi necessari per il trattamento dei file in ambiente AmigaDOS. È bene eliminare tutti i comandi che non servono per la programmazione; inoltre, è possibile sveltire le operazioni rendendo residenti i comandi più utili, e sintetizzandone i nomi tramite gli alias.

- La subdirectory t. Contiene tutti i file temporanei creati dal sistema o dagli altri programmi in esecuzione. La maggior parte dei programmi di edit creano i loro file di backup in questa directory. Di solito i file presenti in questa subdirectory servono solo per sicurezza, dato che comprendono l'ultima versione di un programma sorgente appena corretto.

- La subdirectory fonts. Contiene i file delle fonti-carattere. La fonte Topaz è sempre disponibile direttamente nel sistema senza essere presente nella directory fonts. Quindi, se non si prevede di usare altre fonti, quelle presenti nella directory fonts possono essere tutte cancellate.

Il disco del compilatore

Nel disco da inserire nel disk drive aggiuntivo vanno collocate le directory che elenchiamo qui di seguito.

- La directory radice. Contiene tutti i file sorgente che non trovano spazio nel disco sistema. I file sorgente possono trovarsi nel disco sistema, nel disco del compilatore o nel RAM disk. L'unica condizione indispensabile è che i file comandi relativi alle fasi di compilazione e di link facciano riferimento alle subdirectory in cui quei file effettivamente si trovano. Se si dispone di memoria RAM sufficiente, è possibile copiare nel RAM disk i file sorgente e gli altri file richiesti dall'ambiente di programmazione in C: in tal modo, il processo di compilazione viene notevolmente accelerato.

- La subdirectory c. Contiene il compilatore e il linker del linguaggio C. Nel caso del compilatore Lattice, questi file sono chiamati LC1, LC2 e Blink (versione 5.0). LC1 serve nella prima fase di lavoro: utilizza il file sorgente per scrivere un file provvisorio. Quest'ultimo viene poi inviato in input a LC2, che produce un file oggetto, a sua volta utilizzato dal linker Blink per generare un file eseguibile in output, insieme con un file che riporta i vari messaggi di errore generati durante la compilazione e il link. I file usati in questo processo e il file eseguibile risultante vengono automaticamente salvati nella directory dove si trova il file sorgente. Il disco utilizzato per i file sorgente deve quindi riservare loro lo spezio necessario.

- La subdirectory lib. Contiene i file oggetto e i dati richiesti per le due fasi di compilazione dei programmi LC1 e LC2. Inoltre devono esserci anche i file Astartup.obj e Lstartup.obj, il file amiga.lib, il file debug.lib (serve solo per la fase di debug), e il file lc.lib, che contiene funzioni specifiche del compilatore della Lattice. A questi file fa riferimento il file comandi che esegue la compilazione e il link. Il file amiga.lib contiene anche le funzioni di supporto alla libreria Exec di cui parleremo nel capitolo 2. A differenza delle librerie shared, questa è una libreria linked, cioè una di quelle che intervengono solo in fase di compilazione se vengono esplicitamente richiamate. Quindi, quando si sviluppa un programma in C che utilizza funzioni della libreria amiga.lib, non occorre nessuna istruzione di tipo OpenLibrary o OpenDevice.

- La subdirectory fd.files. Contiene file di descrizione necessari ai programmi del compilatore per determinare gli offset dei vettori e delle funzioni.

- La subdirectory include. Contiene tutti i file INCLUDE necessari per la programmazione in C. I file INCLUDE contengono definizioni di strutture, nomi di flag e tutte le costanti che servono per compilare i programmi in C.

Si noti che, generalmente, durante la procedura di startup vengono impartiti dei comandi ASSIGN dell'AmigaDOS allo scopo di associare alle subdirectory finora elencate i necessari dispositivi logici riconosciuti dal sistema. Questa procedura consente al compilatore e all'utente di fare riferimento sempre al medesimo dispositivo logico, anche nel caso che venisse variata la posizione di alcuni file nell'albero delle subdirectory oppure il nome stesso delle subdirectory. Per una spiegazione dettagliata del comando ASSIGN, si consulti il volume *Il manuale dell'AmigaDOS* pubblicato dalla IHT Gruppo Editoriale. Per convenzione, questi dispositivi logici hanno lo stesso nome delle subdirectory, ma scritto in maiuscolo e seguito dai due punti.

I dispositivi di I/O

Introduzione

Questo capitolo analizza le procedure d'accesso ai dispositivi e le interazioni task-dispositivo. Vengono presentate tutte le funzioni e i comandi standard dei 12 dispositivi dell'Amiga, insieme con le appropriate strutture e le informazioni contenute nei file INCLUDE a essi relativi. Molti degli argomenti qui presentati sono estensioni di argomenti discussi nel capitolo 1 del primo volume. Quando i concetti illustrati in questo capitolo saranno stati assimilati si sarà sulla buona strada per comprendere il funzionamento dei dispositivi dell'Amiga. Passo dopo passo si comprenderà come utilizzare i dispositivi predefiniti in modo efficiente, e come aggiungerne di propri al sistema.

Interazioni task-dispositivo

La Figura 1.1 (a pagina 5) descrive le principali interazioni fra un task e un dispositivo. Esistono diverse chiavi d'interpretazione per analizzare questo schema. La prima è comprendere come lavora ogni funzione. Le funzioni sono analizzate in ogni dettaglio nel primo volume, ma quelle che riguardano i dispositivi sono illustrate anche qui. La seconda chiave è capire come agiscono i comandi di ogni dispositivo. La terza chiave è comprendere chiaramente la differenza esistente fra l'I/O accodato (QueuedIO) e l'I/O veloce (QuickIO). La quarta e ultima chiave è capire la differenza esistente fra l'I/O sincrono e quello asincrono.

Si analizzi ora la Figura 1.1 accanto alla Figura 1.2 del primo volume. Si può notare che l'interazione task-dispositivo non è altro che un particolare tipo d'interazione tra task e task, nel quale le routine del secondo task, organizzate all'interno di una libreria, sono predefinite. Il comportamento e le modalità d'impiego delle strutture MsgPort, Message, e dei segnali discussi nel primo volume, sono praticamente gli stessi. Le uniche eccezioni sono le seguenti.

- La gestione dei segnali relativi alle routine del dispositivo viene condotta automaticamente dal dispositivo stesso.

- Le risposte a richieste di I/O sono gestite dalle routine interne del dispositivo utilizzando la funzione ReplyMsg.

- La decisione di elaborare una richiesta di I/O in modo veloce viene presa dalle routine interne del dispositivo. Si può inoltrare una richiesta di I/O richiedendo il QuickIO, ma se il dispositivo non è in grado di soddisfarla viene trattata come una richiesta di I/O accodato.

Analizzando la Figura 1.1 insieme con le definizioni delle strutture

IORequest, IOStdReq, MsgPort, Message, Device e Unit, è possibile capire in che modo un task riesce a comunicare con le singole unità dei dispositivi. Vediamo ora come sono definiti i dispositivi dell'Amiga.

dispositivi di I/O

In pratica i dispositivi sono librerie, come la libreria Intuition e la libreria Graphics. Al pari di una libreria, un dispositivo può trovarsi su ROM, oppure su disco. In questo secondo caso, quando un task chiama la funzione OpenDevice per aprirlo, il sistema controlla se è già presente in memoria, e se non lo è provvede a caricarlo da disco. Per gestire un dispositivo viene impiegata una struttura Device, esattamente uguale alla struttura Library impiegata per gestire le librerie, che il sistema provvede a creare quando deve rendere disponibile un dispositivo (un dispositivo viene definito disponibile quando è presente in memoria). Questa struttura riporta i parametri di gestione del particolare dispositivo, e rimane in memoria finché vi rimane il dispositivo. Fra le informazioni che riporta, tra cui le dimensioni del dispositivo e la sua versione, è presente anche il numero di volte che la libreria è stata aperta dai task senza essere stata chiusa. A ogni unità corrisponde una propria struttura di tipo Unit, che contiene la message port adibita a ricevere le richieste di I/O inoltrate all'unità, e tiene conto del numero di volte che è stata aperta dai task senza essere stata chiusa (un'unità può essere aperta più volte solo se è ad accesso condiviso).

La struttura Device viene creata dal sistema quando rende disponibile il dispositivo, e non dai task. Il sistema può rendere disponibile il dispositivo soltanto in due casi: durante il boot, se il dispositivo è residente su ROM, oppure in seguito a una chiamata alla funzione OpenDevice se il dispositivo è residente su disco e non è ancora stato caricato in memoria. Nell'operazione chiama anche la funzione AddDevice (che sarà illustrata nel prossimo capitolo) al fine d'inserire la struttura Device del dispositivo all'interno della lista di sistema DeviceList individuata dalla struttura ExecBase. In questa lista compaiono infatti le strutture Device di tutti i dispositivi disponibili. Se per esempio un task chiama la funzione OpenDevice indicando il dispositivo Serial, il sistema controlla se nella lista DeviceList è presente la relativa struttura Device di gestione; se non la trova, desume che il dispositivo non è disponibile e provvede a caricarlo da disco. Al contrario, se un dispositivo residente su disco viene rimosso dalla memoria tramite la funzione RemDevice, la sua struttura Device viene estratta dalla lista e rimossa a sua volta.

Per comunicare con i dispositivi è necessario servirsi di particolari messaggi chiamati richieste di I/O. Vediamoli in dettaglio.

Le richieste di I/O: messaggi per i dispositivi

Quando un task ha bisogno d'inviare a un dispositivo una richiesta di I/O, deve servirsi della particolare struttura di I/O richiesta da quel dispositivo;

nella maggior parte dei casi si tratta della struttura IORequest o della più estesa struttura IOStdReq, ma alcuni dispositivi impiegano queste strutture come primo elemento all'interno di strutture non standard più complesse. Per esempio, il dispositivo Serial utilizza la struttura IOExtSer, mentre il dispositivo Parallel utilizza la struttura IOExtPar, che non sono standard, ma possiedono entrambe come primo elemento una sotto-struttura IOStdReq.

Una richiesta di I/O non è altro che un messaggio che dal task viene inviato a una particolare unità. Il messaggio viene elaborato dalle routine interne del dispositivo, e infine viene restituito al mittente. I messaggi costituiscono i mezzi di trasporto delle informazioni che i task, i dispositivi e il sistema si scambiano vicendevolmente. La loro formulazione è generale, cioè vale per l'intero sistema. Il fatto che vengano usati non solo fra task e task, ma anche fra task e dispositivo, mette in evidenza che l'interazione task-dispositivo è per molti aspetti simile a quella task-task vista nel primo volume.

Un messaggio è costituito da una struttura di tipo Message seguita in memoria dai dati che compongono il messaggio vero e proprio. La struttura Message è l'intestazione del messaggio: serve a mantenerlo all'interno delle code delle message port nelle quali verrà accodato (il primo parametro è infatti una struttura di tipo Node, che stabilisce la doppia concatenazione con gli altri nodi della generica lista in cui il messaggio viene inserito), e a individuare la reply port (cioè la message port del mittente) alla quale il messaggio dev'essere restituito dopo essere stato elaborato. Sempre nella stessa struttura viene indicata anche la lunghezza del messaggio (che può arrivare fino a 64K). Talvolta il messaggio è di lunghezza prestabilita, nota sia al mittente sia al destinatario; in questo caso, come accade per i dispositivi dell'Amiga, non è necessario indicarla esplicitamente.

Il mittente viene deciso dal task che prepara il messaggio; normalmente lo fa corrispondere con la sua reply port, ma non è obbligato a seguire questa consuetudine. Non è detto nemmeno che il mittente, una volta stabilito, rimanga invariato: può infatti accadere che il destinatario del messaggio, quando lo riceve, ne cambi il mittente secondo le proprie esigenze. Infine, se chi invia il messaggio non indica alcun mittente, il messaggio non viene restituito.

Nella gestione dei messaggi esiste una regola fondamentale che non bisogna mai dimenticare: quando un task ha inviato un messaggio nel sistema (a un altro task o a un dispositivo), non deve più alterarlo fino a quando non lo riceve sotto forma di risposta. Una volta inviato, infatti, il messaggio diventa di proprietà del destinatario, il quale può elaborarlo, copiarlo in parte o completamente, ed eventualmente alterarlo per restituire dei risultati. Il destinatario, a sua volta, può accedere al messaggio solo finché non lo restituisce al mittente indicato nella struttura Message: nel momento in cui il messaggio viene restituito tramite la funzione ReplyMsg, torna sotto il controllo del mittente, che può alterarlo, impiegarlo nuovamente, o distruggerlo (cioè liberare la memoria da esso occupata).

Un altro concetto che bisogna avere ben chiaro è che il messaggio, quando viene inviato, di fatto continua a occupare sempre gli stessi byte che per esso erano stati allocati nella memoria. Quando si dice che un messaggio viene *inviato* a una message port, significa in realtà che solo l'indirizzo della relativa

struttura Message viene inviato, e non il messaggio. Allo stesso modo, quando si dice che un messaggio viene restituito al mittente, oppure che il mittente riceve la risposta, significa che la struttura del messaggio è tornata di proprietà del mittente, il quale può così accedervi per ottenere i dati restituiti dal destinatario.

Le richieste di I/O ai dispositivi, formulate tramite le strutture di I/O standard (IORequest e IOStdReq) o non standard, sono anch'esse dei messaggi: infatti il primo parametro di queste strutture di I/O è proprio una struttura Message, ovvero l'intestazione di un messaggio. Queste richieste vengono allocate dal task e inizializzate parzialmente dalla funzione OpenDevice che il task chiama per accedere a un particolare dispositivo. Tra le altre mansioni, OpenDevice provvede a memorizzare nella struttura di I/O indicata dal task l'indirizzo della struttura Device relativa al dispositivo e l'indirizzo della struttura Unit relativa all'unità. In seguito, il task impiega questa struttura di I/O per interagire con il dispositivo fino a quando non decide di chiuderlo. Nel primo volume vengono analizzate le definizioni e la gestione delle strutture Library e Device.

Il flusso delle richieste di I/O

La Figura 1.1 (nella pagina successiva) mostra un task (il grande rettangolo sulla sinistra) e un'unità di un dispositivo (il grande rettangolo sulla destra). Questi rettangoli rappresentano istruzioni Assembly in esecuzione nell'Amiga. Il rettangolo del task rappresenta tutte le istruzioni che costituiscono il task, includendo quelle che riguardano specificamente le interazioni task-dispositivo e la preparazione, l'invio e l'elaborazione delle richieste di I/O. In particolare, include le istruzioni che effettuano chiamate a funzioni come OpenDevice, CloseDevice, BeginIO, DoIO, SendIO, AbortIO, CheckIO, WaitIO, Wait, Remove, WaitPort e GetMsg (in questo capitolo ci limitiamo a citare le varie funzioni, e ne rimandiamo la descrizione completa al secondo capitolo). Nei codici del task sono inoltre presenti tutte le istruzioni che definiscono i parametri delle strutture necessarie per inviare ciascuna richiesta di I/O. Se viene usata la struttura IOStdReq, significa che la richiesta di I/O prevede lo scambio di una certa quantità di dati; infatti, uno dei parametri più importanti di questa struttura, non presente in IORequest, è io_Data, che indica il buffer definito dal task per scambiare dati con il dispositivo.

Il grande rettangolo di destra, invece, rappresenta le routine interne che l'unità impiega per svolgere i propri compiti (le routine del dispositivo sono condivise da tutte le sue unità). Se l'unità è ad accesso condiviso, tutti i task che ne ottengono l'accesso condividono in pratica le stesse routine del dispositivo. Nella maggior parte dei dispositivi, ogni unità possiede una serie di buffer interni che utilizza per elaborare le richieste di I/O provenienti dai task. Questi buffer sono definiti e controllati dalle routine interne del dispositivo, e non dai task. Rappresentano locazioni intermedie per il transito di dati in partenza o in arrivo tra l'unità e altre aree del sistema (in particolare l'hardware esterno).

La request port dell'unità (la message port dell'unità che riceve dai task le richieste di I/O) e la reply port del task (la message port del task che riceve le risposte inoltrate al mittente dalle unità) sono rappresentate nella Figura 1.1 da due rettangolini sotto i due rettangoli più grandi. Si tratta di due message port utilizzate per gestire la coda di messaggi in arrivo all'unità (coda alla request port) e la coda di messaggi in arrivo al task (coda alla reply port). Queste code sono liste a doppia concatenazione che elencano le richieste di I/O in attesa di giungere alla sommità per essere elaborate. Le richieste di I/O sono rappresentate nella figura dai rettangoli ancora più piccoli disposti sotto le message port.

Il dispositivo possiede generalmente una coda per ognuna delle sue unità. A sua volta, il task può disporre di un numero qualsiasi di reply port, e quindi di code per le risposte. La differenziazione delle reply port può servire a un task quando si aspetta di ricevere diverse classi di dati dalla stessa unità.

Quando un task invia una richiesta a un'unità, la routine interna BeginIO del dispositivo (che interviene sempre in ogni interazione con qualsiasi dispositivo) rileva a quale unità è diretta e la inserisce nella corrispondente coda. La coda alla request port dell'unità mantiene in attesa tutte le richieste di I/O provenienti dai vari task del sistema. Le code alla reply port del task contengono invece le risposte inviate dai dispositivi.

I task possono intervenire sulla gestione della coda alla request port

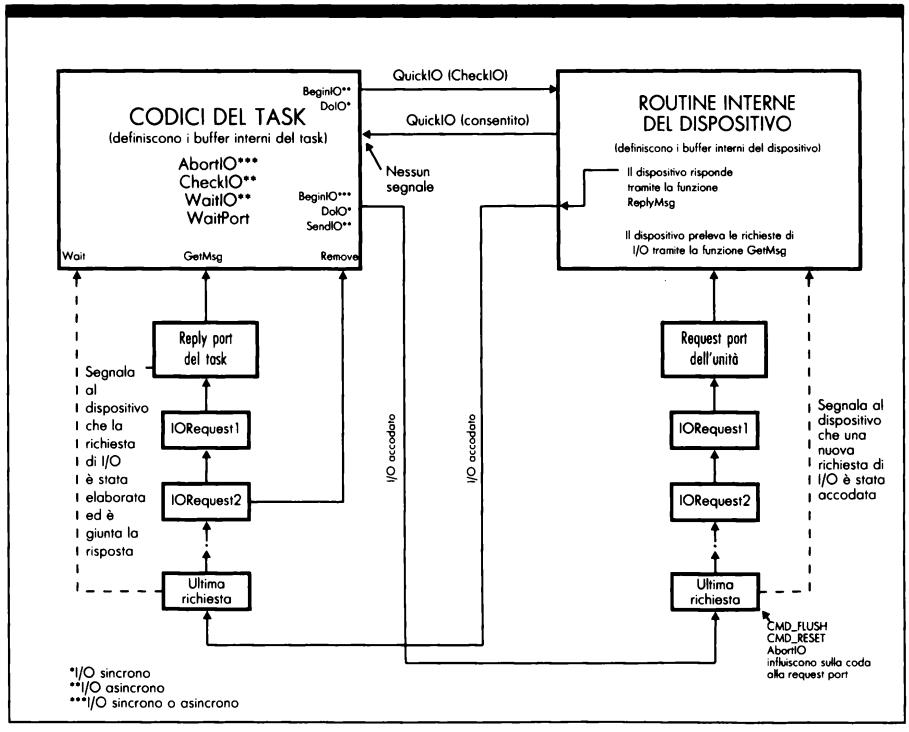


Figura 1.1: *Interazioni fra task e dispositivo tramite le strutture per le richieste di I/O*

dell'unità tramite due comandi standard comuni a tutti i dispositivi (CMD_FLUSH e CMD_RESET, che rimuovono tutte le richieste presenti nella coda) e una funzione della libreria Exec (AbortIO, che invece rimuove una particolare richiesta).

Le linee nella figura descrivono il flusso d'informazioni tra un task definito dal programmatore e le routine interne del dispositivo. Si consideri per prima la linea che procede dal task fino alla parte inferiore della coda alla request port: essa individua il cammino svolto dalle richieste di I/O accodato che il task, tramite le funzioni BeginIO, DoIO e SendIO, invia all'unità.

La linea che procede dall'unità fino alla parte inferiore della coda alla reply port del task indica invece il percorso delle risposte alle richieste di I/O restituite dall'unità tramite la funzione ReplyMsg. Si tratta sempre di richieste di I/O che hanno raggiunto la sommità della request port, sono state elaborate e successivamente restituite. Si ricordi che essendo la richiesta di I/O un messaggio, la relativa risposta utilizza la stessa struttura della richiesta, all'interno della quale uno o più parametri possono essere stati aggiornati per documentare l'esito della richiesta ed eventualmente per restituire altri dati.

La linea appena più in alto proveniente dal task rappresenta le richieste di I/O che nelle loro strutture di I/O possiedono il flag IOF_QUICK impostato. Queste richieste sono intese come richieste di I/O veloce, ma nel nostro esempio il dispositivo non è in grado di gestirle in quanto tali, e di conseguenza vengono trattate come normali richieste di I/O accodato a cui l'unità risponde utilizzando la funzione ReplyMsg.

Classi delle richieste di I/O

Le richieste di I/O ai dispositivi si dividono fondamentalmente in due classi: richieste di I/O accodato e di I/O veloce. Una terza classe di richieste, per le quali vengono utilizzati comandi "in modo immediato", verrà analizzata nel capitolo 2.

I/O accodato

Nell'I/O accodato, il task invia a una specifica unità di un dispositivo una richiesta. La richiesta viene inserita dal dispositivo nella coda alla request port dell'unità indirizzata. Le richieste di I/O presenti nella coda vengono elaborate dall'unità via via che raggiungono la sommità della coda.

Lo svuotamento della lista di richieste viene eseguito automaticamente dalle routine interne del dispositivo grazie a una funzione analoga a GetMsg, della libreria Exec. Le routine interne accedono alla richiesta di I/O che si trova in cima alla coda impiegando l'indirizzo che ottengono da questa funzione, la quale provvede anche a rimuovere la richiesta dalla lista. Successivamente, i dati della richiesta vengono elaborati e restituiti al mittente tramite la funzione ReplyMsg, la quale provvede a inserire la struttura di I/O nella coda alla reply port del task. Quando la risposta giunge alla sommità di questa seconda coda,

il task vi accede per leggere i dati restituiti dal dispositivo.

I modi in cui il task può inviare una richiesta di I/O accodato e ricevere una risposta sono due, a seconda che venga impiegato l'I/O asincrono (richieste inviate tramite le funzioni SendIO o BeginIO) o l'I/O sincrono (richieste inviate tramite le funzioni DoIO o BeginIO; si noti che la funzione BeginIO può autonomamente servirsi sia dell'I/O sincrono sia dell'I/O asincrono). Su questi due tipi di I/O ci soffermeremo più dettagliatamente nel prossimo capitolo; per ora ci limiteremo a una trattazione generica.

Se il task ha inviato una richiesta di I/O asincrono tramite le funzioni SendIO o BeginIO, riottiene subito il controllo, in qualche caso prima ancora che l'unità abbia avuto il tempo di riceverla. Per sapere quando è pronta la risposta deve preoccuparsi di verificare periodicamente, tramite la funzione CheckIO, la propria reply port: se la risposta è pervenuta ne ottiene l'indirizzo da CheckIO e procede a chiamare la funzione Remove per estrarla dalla coda alla reply port (Remove estrae un nodo da una lista in qualunque punto si trovi). La funzione CheckIO non blocca l'esecuzione del task: se l'esito è negativo, il task può svolgere altre mansioni e rieffettuare il controllo in un secondo momento.

Al posto di CheckIO, il task potrebbe anche chiamare ripetutamente la funzione GetMsg, la quale restituisce l'indirizzo della struttura di I/O che si trova alla sommità della coda indicata. Questo secondo metodo si usa in genere quando il task si aspetta di ricevere alla sua reply port una varietà di messaggi, provenienti non solo da una particolare unità, ma anche da altri task o da altre unità. Per distinguere i messaggi in arrivo e riconoscere la risposta alla richiesta, il task può confrontare ogni indirizzo che ottiene da GetMsg con l'indirizzo della struttura di I/O che ha impiegato per inviare la richiesta.

Oltre a questi due metodi, il task, sempre nel caso della richiesta di I/O asincrono, può chiamare la funzione WaitIO per attendere la risposta nella sua reply port. Questa funzione, a differenza di CheckIO, "addormenta" il task finché non arriva la risposta. Quando questa giunge, WaitIO restituisce un codice d'errore e provvede a rimuoverla dalla coda. Se si usa WaitIO non si ottiene nessun indirizzo, ma il task non ne ha bisogno in quanto l'indirizzo della richiesta che ha inviato è ovviamente quello che ha indicato come parametro della funzione stessa. A questo proposito bisogna sempre ricordarsi che se si usano le funzioni CheckIO o WaitIO, occorre mantenere sempre una copia dell'indirizzo della richiesta di I/O. Si noti infine che chiamare SendIO e subito dopo WaitIO è come chiamare la funzione DoIO, con l'unica differenza che non si richiede il QuickIO.

Se invece il task ha inviato una richiesta di I/O sincrono, utilizzando BeginIO o DoIO, sono queste stesse funzioni ad attendere che il dispositivo abbia completamente elaborato la risposta. BeginIO funziona in modo sincrono solo se il task richiede esplicitamente il QuickIO e il dispositivo lo accoglie, altrimenti il suo funzionamento diventa asincrono e la risposta giunge alla reply port del task. Invece, DoIO richiede sempre il QuickIO, e se questo non viene accordato provvede a rimuovere automaticamente la risposta alla richiesta quando questa arriva alla reply port del task; il suo comportamento è pertanto sempre sincrono. L'I/O sincrono viene normalmente impiegato quando un task non può svolgere altre operazioni se non riceve la risposta alla sua richiesta.

Tutte le richieste di I/O accodato possono infine provocare l'invio di

segnali al task, nel momento in cui la loro elaborazione è stata completata e sono state inserite nella coda alla reply port del task. A questo proposito, illustriamo brevemente i segnali, ampiamente discussi nel primo volume, in riferimento alle risposte che provengono dai dispositivi.

Il meccanismo di transito dei segnali è gestito attraverso la struttura MsgPort usata dal task come reply port. Vediamo un caso pratico: nel parametro mp_Flags della struttura MsgPort è stato indicato che all'arrivo di un messaggio nella coda la message port deve generare un segnale, nel parametro mp_SigTask è stato inserito l'indirizzo della struttura Task corrispondente al task da avvisare con un segnale, e nel parametro mp_SigBit è presente il numero del bit di segnale che all'arrivo del messaggio dev'essere impostato. In queste condizioni, quando nella coda viene inserito un nuovo messaggio il task indicato viene riattivato. Perché questo avvenga bisogna che il task sia in attesa di quel segnale, cioè che abbia precedentemente chiamato la funzione Wait, la quale non restituisce il controllo fino a quando non riceve uno dei segnali indicati come argomento. Quando il task riottiene il controllo può rilevare quale message port ha generato il segnale verificando quale bit di segnale si trova impostato nella propria struttura Task. È importante notare che quando un task è in attesa di uno o più segnali tramite la funzione Wait, non ottiene alcun ciclo della CPU.

I/O veloce

La seconda più importante classe di I/O è l'I/O veloce (a cui ci riferiamo spesso con il nome QuickIO). Se il dispositivo accorda il QuickIO, la richiesta non viene accodata alla request port dell'unità indirizzata, e viene elaborata subito. Per ottenere il QuickIO, il task può chiamare BeginIO, e allora deve impostare il flag IOF_QUICK contenuto nel parametro io_Flags della struttura di I/O, oppure può chiamare DoIO la quale richiede il QuickIO automaticamente; in ambedue i casi il dispositivo, se le condizioni lo consentono, elabora la richiesta immediatamente.

Se la richiesta di I/O veloce viene accolta, dopo l'elaborazione nessuna risposta viene inserita dal dispositivo nella coda alla reply port del task, e non viene inviato al task nessun segnale. Se invece il dispositivo non è in grado di trattare la richiesta come QuickIO, questa viene considerata come qualunque altra richiesta accodata, e il flag IOF_QUICK viene azzerato.

Il task può verificare se la richiesta di I/O veloce è stata accolta leggendo lo stato del flag IOF_QUICK quando riottiene il controllo da DoIO o da BeginIO: se il flag risulta ancora impostato, significa che il QuickIO è stato accordato. Ogni dispositivo decide autonomamente quando è possibile accordare l'I/O veloce, basandosi sulle condizioni in cui si trova il sistema in quel particolare momento. Se il dispositivo non è impegnato. di solito accetta le richieste di I/O veloce.

La funzione SendIO, quando riceve il controllo, azzera sempre il parametro io_Flags della richiesta, non permettendo al task d'impostare il flag IOF_QUICK. Al contrario, la funzione DoIO (sincrona) cerca sempre di ottenere il QuickIO quando invia una richiesta di I/O. Infatti, se il task dopo aver

riottenuto il controllo accede al flag IOF_QUICK, lo trova quasi sicuramente impostato (sempre che il dispositivo abbia potuto accordare il QuickIO). Infine, la funzione BeginIO (sincrona o asincrona, secondo i casi) cerca di ottenere il QuickIO solo se è stato esplicitamente richiesto.

Per i comandi di alcuni dispositivi, le richieste di I/O veloce vengono sempre trasformate in richieste accodate. Nel corso del libro verrà evidenziato per ogni comando se si può tentare la via dell'I/O veloce oppure no.

Interazioni con più di una reply port e con più di un'unità

Questa sezione spiega, con l'aiuto di due figure, il modo in cui vengono gestite contemporaneamente più reply port di uno stesso task, e più unità dello stesso dispositivo.

Un task con più reply port

La Figura 1.2 mostra un task che interagisce con una sola unità di un dispositivo, ma che utilizza un certo numero di reply port per gestire differenti tipi di dati provenienti dal dispositivo. Questa configurazione può essere utile se, per esempio, si sta lavorando con il dispositivo Serial e si desidera suddividere in categorie diverse i dati provenienti da un computer collegato all'Amiga.

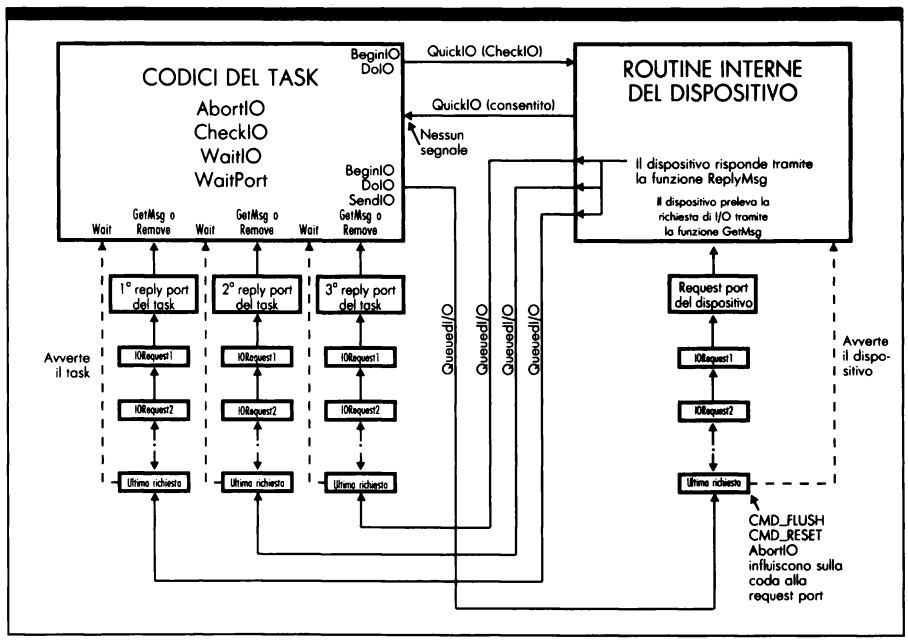


Figura 1.2: *Interazioni fra task e dispositivo quando il task dispone di tre reply port*

Ancora una volta i due rettangoli più grandi rappresentano i codici del task e le routine interne del dispositivo. Immediatamente sotto al rettangolo del task sono rappresentate tre reply port e le rispettive code di attesa delle risposte. A mano a mano che il dispositivo porta a termine l'elaborazione delle richieste di I/O relative a una particolare categoria di dati, la corrispondente reply port riceve risposte appartenenti alla stessa categoria.

I task possiedono un completo controllo sulle risposte e su quali reply port le riceveranno. Ogni task, infatti, quando prepara una richiesta di I/O, deve definirne il mittente memorizzando nel parametro mn_ReplyPort della struttura Message che costituisce l'intestazione della richiesta l'indirizzo di una delle proprie reply port (si ricordi che una reply port è semplicemente una message port adibita a ricevere risposte, e come tale è definita da una strutture MsgPort). I task possono inoltre indicare un buffer utilizzando il puntatore io_Data presente nella struttura IOStdReq (non presente nella più semplice struttura IORequest). Si noti che a parte la differenza determinata dal parametro mn_ReplyPort, l'interazione task-dispositivo definita dalla Figura 1.2 (nella pagina precedente) segue la stessa logica illustrata nella Figura 1.1 (a pagina 5).

Interazioni di un task con più unità

La Figura 1.3 mostra un task che interagisce con più unità di uno stesso dispositivo. Ogni unità è in grado d'inviare risposte a una o più reply port del task, a seconda delle richieste che riceve.

Per indirizzare con le proprie richieste ognuna delle quattro unità, nelle strutture di I/O il task mantiene inalterato il puntatore alla struttura Device, unica per l'intero dispositivo, e cambia di volta in volta il puntatore alla struttura Unit, diversa per ogni unità. Ovviamente, il task può anche allocare e inizializzare tante strutture di I/O quante sono le unità del dispositivo, anziché usare sempre la stessa.

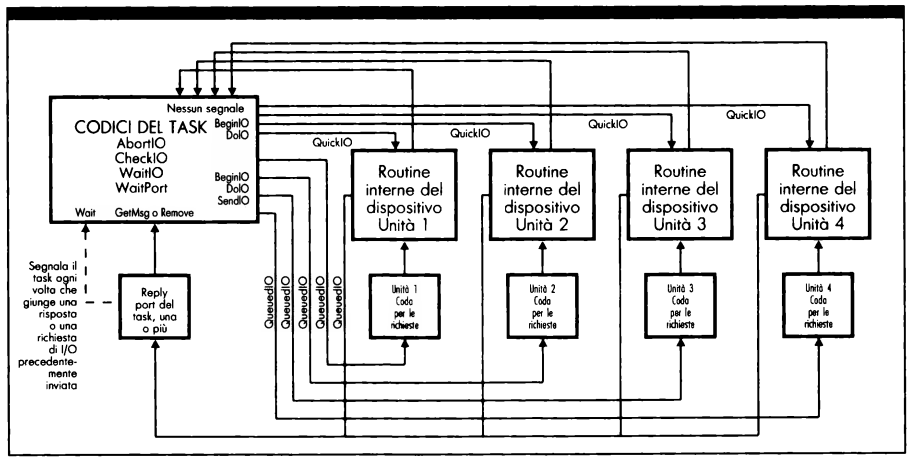


Figura 1.3: *Interazioni tra task e dispositivo nel caso di più reply port e più unità*

La configurazione illustrata è utile se per esempio si sta lavorando contemporaneamente con le quattro unità del dispositivo TrackDisk e si desidera ricevere le risposte alle richieste nella stessa reply port.

Comportamento delle code

Questa sezione spiega, con l'ausilio di due figure, il modo in cui vengono utilizzate nel sistema Amiga le code alle request port delle unità e le code alle reply port dei task.

Comportamento della coda alla request port

La Figura 1.4 illustra come si comporta la coda alla request port di un'unità quando uno o più task vi inseriscono le loro richieste di I/O.

Iniziamo osservando la prima colonna a sinistra che mostra le richieste di I/O in coda alla request port dell'unità. Sono presenti cinque richieste di I/O accodate, ognuna delle quali può provenire da qualsiasi task che abbia aperto questa unità e le abbia inviato una struttura di I/O (IORequest, IOStdReq oppure una struttura di I/O non standard, a seconda del dispositivo). Questi task ovviamente indirizzano nelle loro strutture di I/O le stesse sotto-strutture Device e Unit.

Proseguendo, si osservi lo stato della coda dopo che il dispositivo ha iniziato l'elaborazione della prima richiesta. Ha rimosso IORequest1 dalla coda (tramite una funzione equivalente a GetMsg) e la sta elaborando.

La colonna che segue mostra la condizione della coda dopo che una delle funzioni BeginIO (se il QuickIO non è stato richiesto oppure non è stato accordato), DoIO (se il QuickIO non è stato accordato) o SendIO ha accodato

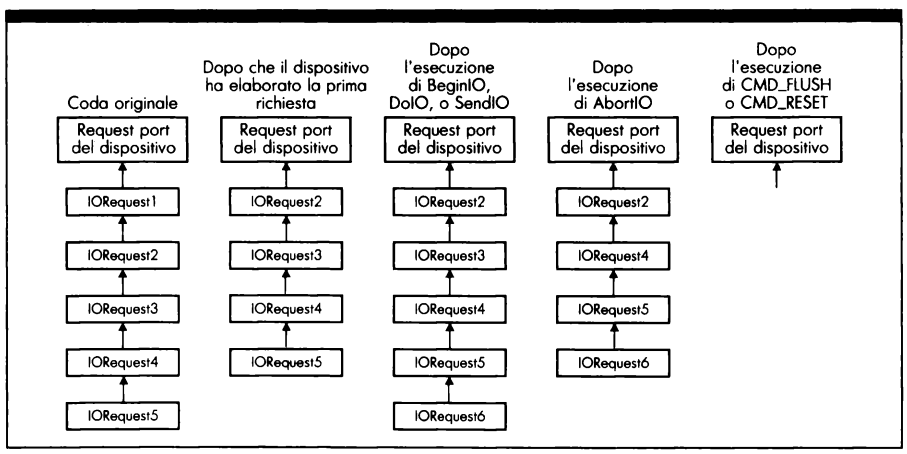


Figura 1.4: *Comportamento della coda a una request port del dispositivo*

un'altra richiesta di I/O (IORequest6) nella coda alla request port dell'unità.

La colonna successiva mostra lo stato della coda dopo che un task ha effettuato una chiamata alla funzione AbortIO per rimuovere una richiesta di I/O non ancora giunta alla sommità della coda e di cui non c'è più necessità. Nel nostro caso è stata rimossa IORequest3. L'ultima colonna mostra infine lo stato della coda successivo all'esecuzione da parte del dispositivo del comando CMD_FLUSH o CMD_RESET (questo comando può essere stato inoltrato da un qualsiasi task, o dal sistema). La coda alla request port è stata svuotata di tutte le richieste di I/O in attesa di essere elaborate, le quali devono essere inoltrate nuovamente se si vuole che il dispositivo possa elaborarle.

Comportamento della coda alla reply port

La Figura 1.5 illustra il comportamento di una reply port quando diverse unità di uno o più dispositivi le inviano le risposte relative alle richieste di I/O di cui hanno concluso l'elaborazione (perché ciò avvenga, nelle definizioni delle richieste dev'essere stato indicato come mittente proprio l'indirizzo di questa reply port). La figura mostra il contenuto della coda alla reply port del task in diversi istanti. Inizialmente essa contiene cinque risposte a richieste di I/O, ciascuna delle quali può essere pervenuta da qualsiasi unità di qualsiasi dispositivo nel sistema.

A fianco della prima situazione è visualizzato lo stato della coda dopo l'acquisizione della prima risposta, IORequest1, da parte del task. Se la richiesta era stata inviata dal task con la funzione SendIO o BeginIO, cioè se si trattava di una richiesta asincrona, il task può utilizzare la funzione GetMsg per ottenere la risposta e rimuoverla dalla coda; tramite GetMsg il task non entra in attesa e ha anche la possibilità di ricevere dalla stessa reply port

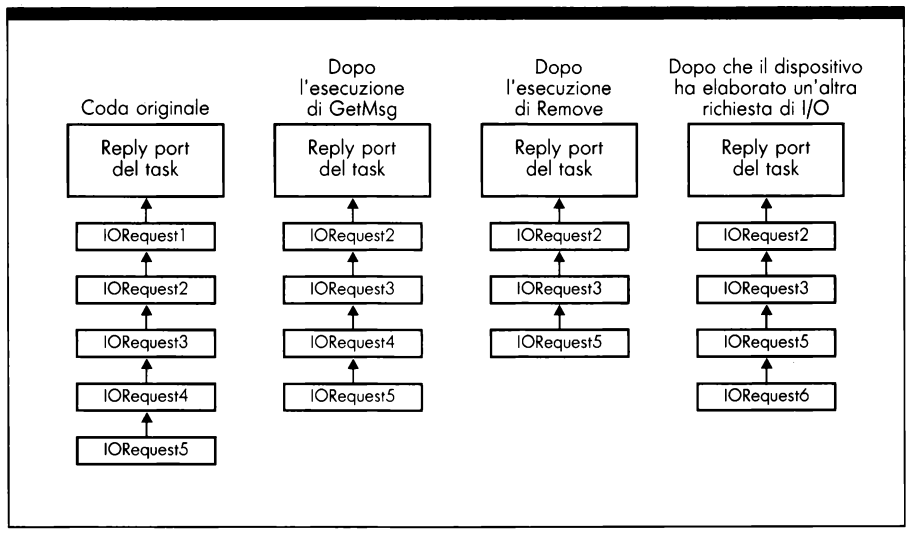


Figura 1.5: *Comportamento della coda a una reply port del task*

messaggi provenienti da mittenti diversi dal dispositivo che ha interpellato, come per esempio altri task. Ma il task che ha effettuato l'invio, anziché GetMsg potrebbe aver chiamato WaitIO: questa funzione mette in attesa il task fino a quando non arriva la risposta, quindi la rimuove dalla reply port e restituisce un codice d'errore.

Se invece la richiesta di I/O è stata inviata tramite la funzione DoIO, cioè si tratta di una richiesta sincrona, e il QuickIO non è stato accordato, questa stessa funzione provvede automaticamente ad attendere e a rimuovere la risposta dalla coda alla reply port, rendendola disponibile al task. Chiamando questa funzione, il task entra ovviamente in attesa.

La successiva colonna della stessa figura mostra la condizione della coda alla reply port dopo che il task ha rimosso IORequest4 tramite la funzione Remove. Per farlo, il task chiama periodicamente la funzione CheckIO al fine di verificare se la particolare risposta (IORequest4) è presente nella coda, e quando ottiene l'indirizzo della struttura di I/O che definisce IORequest4, chiama la funzione Remove per rimuovere la risposta dalla coda. Con questo sistema si accede direttamente a una particolare richiesta scavalcando l'ordinata ascesa all'interno della coda.

L'ultima colonna nella figura mostra infine lo stato della coda dopo che un dispositivo ha portato a termine l'elaborazione di un'altra richiesta di I/O e ha inserito la relativa risposta (IORequest6) nella parte inferiore della coda.

Accesso condiviso ed esclusivo

La Figura 1.6 (nella pagina successiva) mostra la differenza tra il modo di accesso condiviso e quello esclusivo. I dispositivi ai quali si può accedere in tutti e due i modi prendono sempre in considerazione un particolare flag nell'argomento Flags della funzione OpenDevice; questo flag deve infatti indicare il tipo di accesso al dispositivo (e quindi alle sue unità) che i task desiderano operare. Per esempio, il dispositivo Serial al momento dell'apertura controlla lo stato del flag SERF_SHARED per sapere se il task vuole accedere nel modo di accesso condiviso o esclusivo. Si noti che non sempre il modo di default dei dispositivi è quello esclusivo, e inoltre alcuni dispositivi non consentono entrambi i modi.

Nella parte superiore della Figura 1.6 tre task condividono un'unità che hanno aperto indicando l'accesso condiviso, e le inviano le proprie richieste di I/O. Non è necessario che i task si cedano vicendevolmente il controllo dell'unità per inviare le proprie richieste di I/O, infatti, dopo che il primo task ha aperto l'unità tramite la funzione OpenDevice, anche un altro task può aprire la stessa unità e inoltrare le proprie richieste. Non è necessario che Task1 chiuda l'unità prima che Task2 e Task3 possano a loro volta aprirla.

Il modo di accesso esclusivo opera diversamente, come si deduce dalla parte inferiore della Figura 1.6. Task1 deve chiudere l'unità del dispositivo prima che Task2 e Task3 possano accedervi. Questo significa che Task1, dopo aver chiamato la funzione OpenDevice e aver interagito con l'unità tramite le funzioni BeginIO, DoIO oppure SendIO, deve necessariamente chiamare la

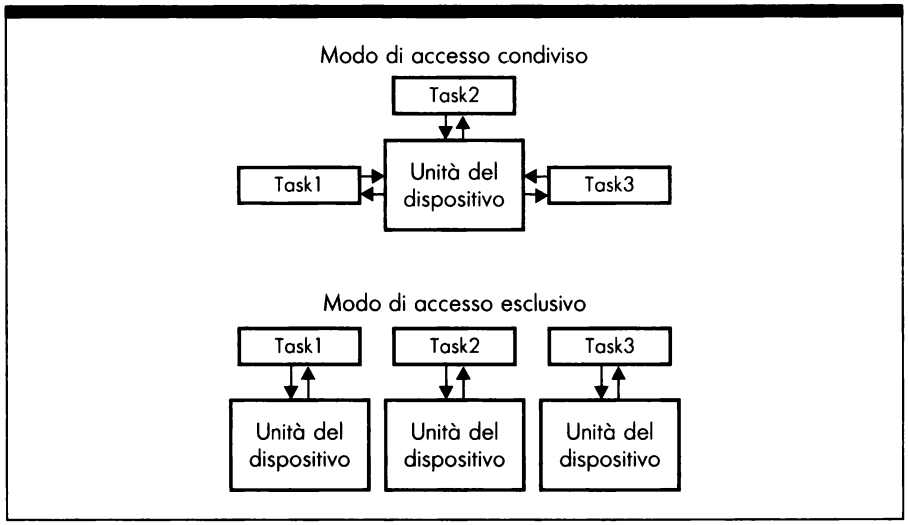


Figura 1.6: *Differenza tra il modo di accesso condiviso e il modo di accesso esclusivo*

funzione CloseDevice perché un altro task possa accedervi.

Questo non significa comunque che lo scambio di controllo fra i task (cioè la gestione multitasking) sia impossibile, ma solo che ogni tentativo da parte di Task2 e Task3 di aprire l'unità prima che Task1 l'abbia chiusa, provoca una condizione d'errore. I task ai quali non viene accordato l'accesso ricevono il codice d'errore IOERR_OPENFAIL, fino a quando il primo task non chiude l'unità. Questi concetti sono basilari nello sviluppo logico di un programma.

Le strutture Device e Unit possiedono un parametro che tiene conto di quante volte il dispositivo e l'unità sono stati aperti e non ancora chiusi. In particolare la struttura Device riporta questa informazione nel parametro lib_OpenCnt e la struttura Unit nel parametro unit_OpenCnt. Il sistema, quando trova lib_OpenCnt a zero (dopo l'esecuzione della funzione CloseDevice), ritiene che il dispositivo sia completamente chiuso ma non la cancella dalla memoria, anche se si tratta di un dispositivo residente su disco. Il sistema prende in considerazione questi due parametri e il tipo di accesso richiesto per decidere quali azioni intraprendere nel momento in cui un task tenta di aprire un'unità di un dispositivo.

Un modo per semplificare queste decisioni è quello di utilizzare tutte le unità disponibili di un dispositivo, ripartendole fra i vari task. In questo modo si evitano facilmente le collisioni che possono nascere tra task che accedono alla stessa unità. Si noti che mentre tutti i dispositivi impiegano la struttura Device, non tutti si servono della struttura Unit, e soprattutto del suo parametro unit_OpenCnt. Queste differenze non sono comunque d'interesse per i programmatori: è sufficiente ricordare che nella struttura di I/O, all'interno dei parametri io_Device e io_Unit, devono memorizzare gli indirizzi che hanno ottenuto aprendo il dispositivo. Questi indirizzi individuano univocamente il dispositivo e la sua unità.

Gestione multitasking ed elaborazione delle richieste di I/O

La Figura 1.7 (nella pagina successiva) illustra nei dettagli le capacità multitasking dell'Amiga quando diversi task inviano una serie di richieste di I/O a una certa unità. Questa figura mostra le diverse elaborazioni delle richieste di I/O asincrono e sincrono.

Tre task (Task1, Task2 e Task3) sono in comunicazione con la stessa unità di un dispositivo. Un esempio tipico potrebbe essere costituito da tre task in comunicazione con il dispositivo Serial, ognuno dei quali tenta di prelevare i propri dati dalla porta seriale dell'Amiga. Task1 deve inviare tre richieste di I/O all'unità del dispositivo, Richiesta 11, Richiesta 12 e Richiesta 13, che rappresentano simbolicamente le strutture di I/O (IORequest, IOStdReq o, per il dispositivo Serial, IOExtSer) utilizzate per definire le richieste. Task2 deve inviare all'unità Richiesta 21, Richiesta 22 e Richiesta 23, mentre Task3 deve inviare Richiesta 31, Richiesta 32 e Richiesta 33.

In questo esempio Task1 possiede la più alta priorità d'esecuzione (ln_Pri = 60), Task2 quella immediatamente successiva (ln_Pri = 55), e Task3 la più bassa (ln_Pri = 50). Il parametro ln_Pri è contenuto nella struttura Node che costituisce in questo contesto il primo parametro della struttura Task relativa a ognuno dei tre task. Ciascuno dei tre task ha aperto la medesima unità del dispositivo tramite la funzione OpenDevice, e ha specificato il modo di accesso condiviso. Infine, la coda dell'unità per le richieste di I/O è attualmente occupata da un certo numero di richieste accodate precedentemente da altri task nel sistema.

I tre task passano attraverso le seguenti fasi.

1. Task1 utilizza la funzione DoIO (che permette l'invio di richieste di I/O sincrono) per inoltrare Richiesta 11. Dal momento che il QuickIO non viene accordato e che la coda all'unità non è vuota, l'unità del dispositivo non è in grado di servire immediatamente la richiesta e quindi il dispositivo la accoda dietro le altre. A causa di questo accodamento, la funzione DoIO non può restituire il controllo a Task1, che entra quindi in attesa. Il sistema procede ad attivare il task con la priorità immediatamente inferiore, ovvero Task2.

2. Task2 ottiene il controllo della CPU e invia alla stessa unità la richiesta di I/O sincrono Richiesta 21, utilizzando la funzione DoIO. Dal momento che il QuickIO non viene accordato, la richiesta viene accodata dietro Richiesta 11 e le altre già presenti nella coda. Task2 entra in stato di attesa.

3. Task3, al livello di priorità più basso, ottiene il controllo della CPU e invia sempre alla stessa unità la richiesta di I/O sincrono Richiesta 31, utilizzando ancora la funzione DoIO. Il QuickIO non viene accordato neanche in questo caso, e la richiesta viene accodata dietro alle precedenti. Ora Task3 risulta anch'esso in stato di attesa.

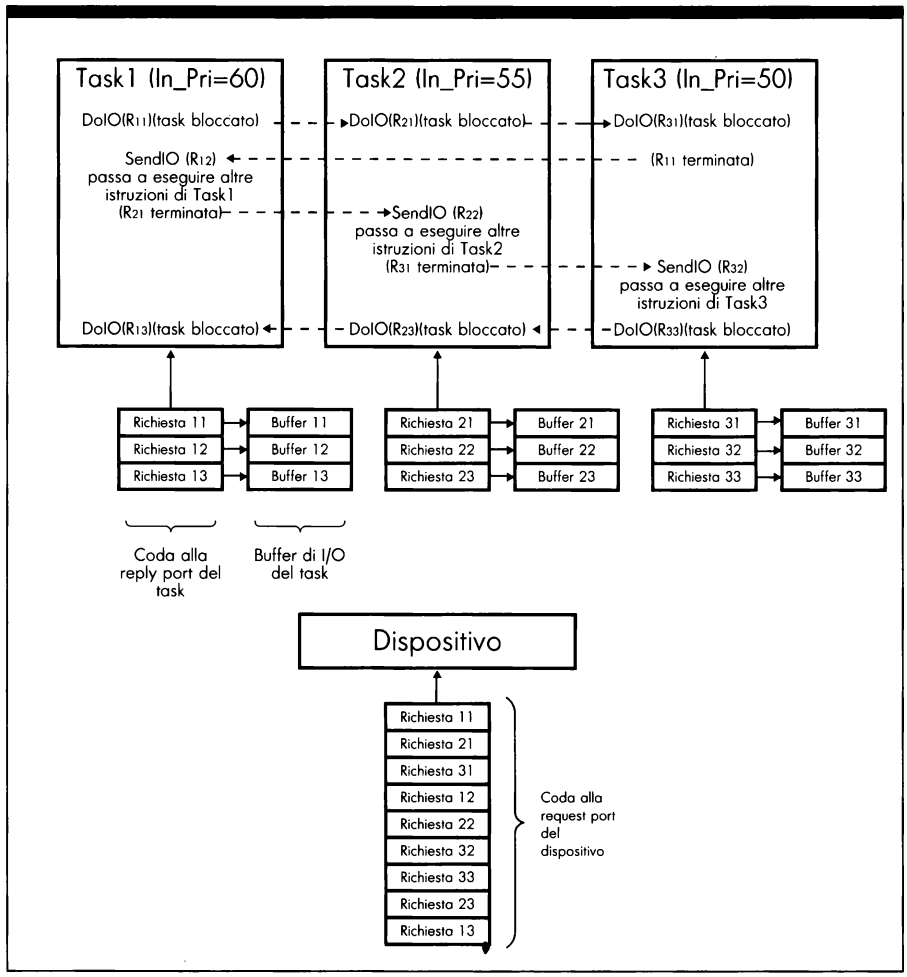


Figura 1.7: *Gestione multitasking delle richieste di I/O*

Si supponga ora che le routine interne del dispositivo abbiano finito l'elaborazione di Richiesta 11 (questa è soltanto un'ipotesi sulla sequenza e la temporizzazione degli eventi nel sistema e non qualcosa che si possa realmente controllare; con quest'ipotesi si suppone anche che siano state elaborate tutte le altre richieste che precedevano Richiesta 11). Il dispositivo restituisce al mittente Richiesta 11 sotto forma di risposta. Task1, precedentemente bloccato, riprende quindi l'esecuzione non appena le routine interne del dispositivo gli segnalano (utilizzando una funzione equivalente a ReplyMsg) che la sua richiesta è stata elaborata. Task1 riceve la risposta nella coda alla propria reply port e DoIO, prima di cedere il controllo, la rimuove dalla coda. Lo scambio di controllo fra i task è indicato con le linee tratteggiate tra i rettangoli che rappresentano i task.

Si supponga ora che il successivo comando in Task1 sia la funzione SendIO.

Questa funzione invia alla request port dell'unità Richiesta 12 (asincrona), che viene accodata dietro Richiesta 21 e Richiesta 31. Dal momento che la funzione SendIO invia richieste di I/O asincrono, Task1 può ora continuare nell'esecuzione di altre istruzioni; periodicamente si preoccuperà di verificare se l'unità ha risposto alla sua richiesta. Intanto l'unità del dispositivo ha finito l'elaborazione di Richiesta 21 (questa è un'altra ipotesi arbitraria), e la restituisce al mittente. Task2 riottiene quindi il controllo della CPU, come mostra la linea tratteggiata che unisce i rettangoli Task1 e Task2. Supponiamo che anche in questo caso la successiva istruzione eseguibile in Task2 sia una funzione SendIO: il processo continua nello stesso modo descritto per Task1.

Se si analizza il diagramma insieme con le spiegazioni relative a DoIO e SendIO (si veda il capitolo 2), risulterà più facile capire come vengono gestite le richieste sincrone e asincrone nel sistema Amiga. Queste considerazioni hanno un considerevole peso nella progettazione dei programmi e dei singoli task di cui sono composti.

dispositivi dell'Amiga

La Figura 1.8 (nella pagina successiva) mostra le relazioni tra i task e i 12 dispositivi predefiniti del sistema Amiga. Il grande rettangolo simboleggia un qualsiasi task del sistema. Le istruzioni contenute all'interno del task includono quelle necessarie per comunicare con i dispositivi, come OpenDevice, CloseDevice, BeginIO, DoIO, SendIO, AbortIO, WaitIO, CheckIO, WaitPort, GetMsg, Remove e Wait, che interagiscono con le request port delle unità dei dispositivi, o con le reply port del task.

I dodici rettangoli più piccoli rappresentano i dispositivi e le loro unità. All'interno di ogni rettangolo è indicato il numero di unità possedute dal dispositivo. Si noti che la libreria Translator, sebbene non sia un dispositivo, viene anch'essa rappresentata con un rettangolo in quanto interagisce direttamente con i dispositivi Audio e Narrator. Gli aspetti più importanti che la Figura 1.8 riassume sono i seguenti.

- Con sufficiente memoria disponibile, ogni task può aprire fino a 12 dispositivi predefiniti, ottenendo l'accesso in simultanea con altri task tramite il modo di accesso condiviso. Un task può inoltre aprire più di un'unità per ognuno dei 12 dispositivi: in effetti, può aprire tutte le unità di tutti i dispositivi. La principale limitazione è costituita dalla memoria. Se vengono aperte tutte le unità di tutti i dispositivi, un elevato numero di richieste di I/O verranno accodate e occuperanno molta memoria RAM.

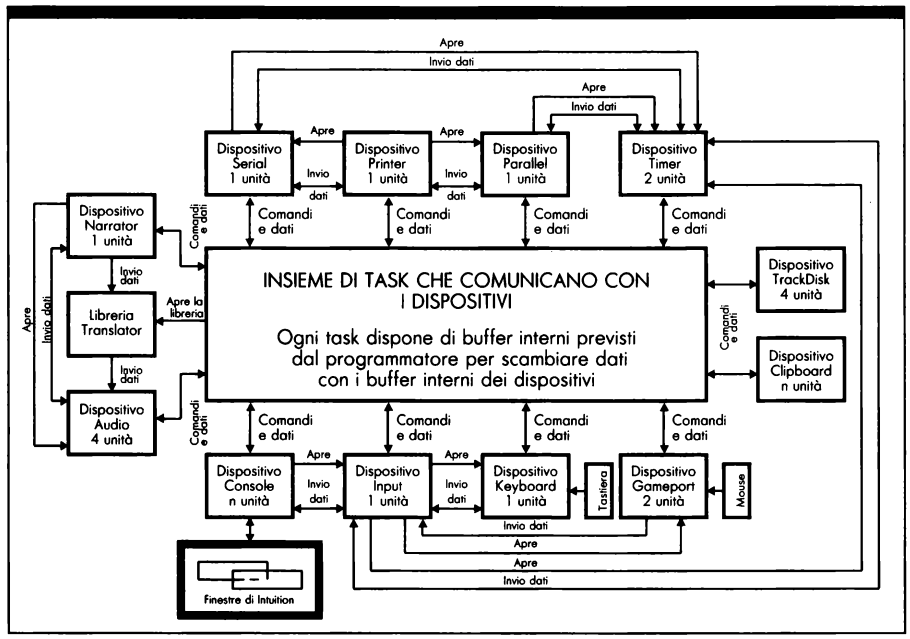


Figura 1.8: *Interazioni fra task e dispositivo per tutti i dispositivi dell'Amiga*

- Le frecce a due sensi che si diramano dal grande rettangolo e arrivano fino ai rettangoli dei dispositivi rappresentano le interazioni tra task e dispositivo; il trasferimento dei comandi e dei dati tra il task e le routine interne del dispositivo viene effettuato dalle funzioni chiamate dall'interno del task. In particolare queste frecce rappresentano le chiamate alle funzioni OpenDevice, CloseDevice, BeginIO, DoIO e SendIO.

- Le frecce etichettate con Apre e Invio dati descrivono le operazioni interne e individuano le relazioni fra i dispositivi. Per esempio dal dispositivo Console parte una freccia etichettata con la parola Apre che arriva fino al dispositivo Input. Questo significa che, dopo essere stato aperto, il dispositivo Console provvede a sua volta ad aprire automaticamente il dispositivo Input. Le frecce etichettate con Invio dati hanno un significato simile: quando il dispositivo Console apre il dispositivo Input, quest'ultimo può a sua volta inviargli dati. Questi trasferimenti di dati vengono gestiti automaticamente dalle routine interne dei dispositivi.

Conviene analizzare attentamente l'intreccio di relazioni illustrato dalla Figura 1.8. Nei prossimi capitoli verranno discusse una per una.

comandi standard dei dispositivi

La Tavola 1.1 (a pagina 21) riassume i comandi standard previsti per ciascun dispositivo (al massimo nove). Si noti che il numero dei comandi standard previsti varia da dispositivo a dispositivo. Ne descriviamo le caratteristiche principali, rimandando all'analisi di ogni singolo dispositivo per informazioni più approfondite.

- CMD_CLEAR azzera tutti i buffer interni del dispositivo relativi a una particolare unità. Si ricordi che alcuni dispositivi possiedono una serie di buffer interni che utilizzano per gestire i dati in transito verso l'hardware esterno e i task. Ovviamente, CMD_CLEAR non ha alcun effetto sui buffer definiti dai task.

- CMD_FLUSH ordina al dispositivo di abortire tutte le richieste di I/O che sono ancora in attesa nella coda di una particolare unità. Una volta che le richieste sono state rimosse, i task devono reinoltrarle se desiderano che vengano elaborate. Le richieste ritornano ai relativi mittenti con il codice d'errore IOERR_ABORTED.

- CMD_INVALID. Quando i dispositivi ricevono questo comando in genere restituiscono il codice d'errore standard IOERR_NOCMD per indicare che il comando richiesto non è previsto dal dispositivo.

- CMD_READ ordina al dispositivo di leggere un certo numero di byte dai suoi buffer interni e di allocarli nel buffer definito dal task. Il numero dei byte da leggere in genere viene specificato dal task nel parametro io_Length della struttura IOStdReq; il numero di byte che invece vengono effettivamente letti viene restituito dal dispositivo nel parametro io_Actual della stessa struttura. Vi sono comunque alcune eccezioni, e i dettagli che definiscono l'impiego di questo comando variano da dispositivo a dispositivo.

- CMD_RESET ordina al dispositivo di operare il reset di una particolare unità. Le routine interne del dispositivo vengono completamente reinizializzate, ripristinando le condizioni di default. CMD_RESET, inoltre, chiama CMD_FLUSH per abortire tutte le richieste di I/O accodate alla request port dell'unità, e chiama AbortIO per eliminare l'eventuale richiesta in corso di elaborazione. Inoltre, cancella tutte le strutture di dati utilizzate dalle routine interne del dispositivo per l'unità, e ogni registro hardware coinvolto.

- CMD_START ordina che riprenda l'elaborazione dei comandi da parte di un'unità precedentemente bloccata con il comando CMD_STOP. Se quando è stato inviato il comando CMD_STOP l'unità stava elaborando una richiesta, inoltrando CMD_START il comando riprende la propria esecuzione nel punto in cui era stato bloccato. Quando non può farlo, è

il sistema a scegliere il punto da cui il comando deve ripartire.

- CMD_STOP ordina al dispositivo di sospendere l'elaborazione dei dati in una sua particolare unità. L'interruzione avviene non appena è possibile. Tutte le richieste di I/O continuano ad accodarsi, ma l'unità non può elaborarle. La coda alla request port può in questo caso crescere rapidamente, producendo un enorme consumo di memoria da parte delle strutture di I/O che vengono accodate ma non elaborate. Il comando risulta utile per quei dispositivi che richiedono l'intervento dell'utente (per esempio stampanti, plotter e reti di comunicazione).

- CMD_UPDATE ordina al dispositivo di riversare nell'hardware il contenuto di tutti i buffer interni dell'unità indirizzata. Le informazioni mantenute in questi buffer in genere hanno origine nei buffer definiti dai task: i buffer interni dei dispositivi rappresentano punti di momentaneo parcheggio per le informazioni. Di solito quindi il dispositivo effettua quest'operazione automaticamente, come parte dei suoi compiti di routine; tuttavia a volte può essere utile provocare esplicitamente un aggiornamento dei dati sotto il controllo del task creato dal programmatore. Questo controllo diretto è necessario a volte con quei dispositivi che mantengono buffer interni di dati (cache), come i disk drive.

- CMD_WRITE ordina al dispositivo di trasferire un certo numero di byte da un buffer definito dal task nel buffer interno dell'unità, ed eventualmente a un dispositivo hardware esterno (per esempio un disk drive). Il numero di byte viene specificato dal task nel parametro io_Length della struttura IOStdReq; il sistema indica poi il numero di byte effettivamente trasferiti nel parametro io_Actual della stessa struttura, che viene restituita come risposta. Ancora una volta, i dettagli che definiscono l'impiego di questo comando variano da dispositivo a dispositivo.

Dei 12 dispositivi dell'Amiga, quattro sono residenti su disco (Narrator, Parallel, Printer e Serial), mentre gli altri otto sono residenti su ROM. In aggiunta ai comandi standard mostrati nella Tavola 1.1 (nella pagina successiva), la maggior parte dei dispositivi sono programmati per eseguire un certo numero di comandi specifici.

Le funzioni dei dispositivi

L'Amiga fornisce nella libreria Exec nove funzioni standard per ciascun dispositivo (AddDevice, RemDevice. OpenDevice, CloseDevice, DoIO, SendIO, CheckIO, WaitIO, AbortIO). ma non sono le uniche disponibili. Alcuni dispositivi utilizzano altre funzioni della libreria Exec.

Tutti i dispositivi richiedono un esplicita chiamata alla funzione OpenDe-

Tavola 1.1: *Comandi standard previsti dai dispositivi dell'Amiga*

	COMANDI STANDARD								
Dispositivo	CLEAR	FLUSH	INVALID	READ	RESET	START	STOP	UPDATE	WRITE
Audio	✓	✓	–	✓	✓	✓	✓	✓	✓
Clipboard	–	–	–	✓	✓	–	–	✓	✓
Console	✓	–	–	✓	–	–	–	–	✓
Gameport	✓	–	–	–	–	–	–	–	–
Input	–	✓	–	–	✓	✓	✓	–	–
Keyboard	✓	–	–	–	✓	–	–	–	–
Narrator	–	✓	–	✓	✓	✓	✓	–	✓
Parallel	✓	✓	–	✓	✓	✓	✓	–	✓
Printer	–	✓	✓	–	✓	✓	✓	–	✓
Serial	✓	✓	–	✓	✓	✓	✓	–	✓
Timer	–	–	–	–	–	–	–	–	–
TrackDisk	✓	–	–	✓	–	–	–	✓	✓

vice per essere aperti. Inoltre, il dispositivo Keyboard viene sempre aperto automaticamente dal dispositivo Input, il quale a sua volta viene aperto automaticamente dal dispositivo Console che a sua volta viene aperto automaticamente dal sistema nella fase di avvio della macchina. Si noti che il dispositivo Console può essere aperto solamente se è attivo l'AmigaDOS.

Al momento dell'accensione, il sistema attiva automaticamente un task di input residente su ROM. Questo task viene impiegato sia dal dispositivo Console sia da Intuition, che "cattura" gli eventi di input provocati dall'utente quando interagisce con le finestre attraverso il mouse e la tastiera.

Tutti i dispositivi prevedono un'esplicita chiamata alla funzione CloseDevice per essere chiusi. Tuttavia, il dispositivo Console viene chiuso dal sistema in seguito a un reset o a un'interruzione dell'alimentazione.

Una volta comprese le relazioni tra le funzioni standard e le funzioni specifiche, si può procedere alla programmazione.

Le strutture che intervengono nei rapporti fra task e dispositivi

La Figura 1.9 (nella pagina successiva) mostra le relazioni esistenti tra le varie strutture che sono direttamente coinvolte nella gestione delle unità dei dispositivi. Oltre alle strutture standard che presentiamo, alcuni dispositivi prevedono anche specifiche strutture proprie, non comuni ad altri dispositivi.

La struttura IOStdReq contiene la sotto-struttura IORequest (di fatto, la struttura IOStdReq è un'estensione standard della struttura IORequest, e come tale la comprende). Questo significa che all'interno della struttura IOStdReq, compare un'intera struttura IORequest come primo elemento. Se il task necessita di un buffer per comunicare con un dispositivo, deve utilizzare al posto della più semplice struttura IORequest la struttura IOStdReq, la quale include il puntatore io_Data. Per le operazioni di lettura, io_Data dev'essere impostato dal task in modo che individui un buffer nel quale le routine interne del dispositivo possano trasferire i dati ottenuti dal dispositivo hardware. Per le operazioni di scrittura, invece, il task deve memorizzare in io_Data l'indirizzo del buffer nel quale ha inserito i dati da inoltrare al dispositivo.

La struttura IORequest contiene il puntatore io_Device per individuare in memoria la struttura di tipo Device relativa al dispositivo con il quale s'intende dialogare. La libreria Exec utilizza il valore contenuto in questo parametro per localizzare il dispositivo. Questa struttura è identica alla struttura Library, e viene utilizzata per gestire l'intero dispositivo. Il parametro io_Device viene inizializzato dal sistema quando il task chiama la funzione OpenDevice per aprire il dispositivo.

La struttura IORequest contiene inoltre il puntatore io_Unit, che individua in memoria la struttura Unit utilizzata per gestire una particolare unità del dispositivo, il quale si serve del valore di questo parametro per rilevare a quale delle sue unità è diretta la richiesta. Il parametro io_Unit viene anch'esso inizializzato dal sistema quando la funzione OpenDevice restituisce il controllo al task.

Oltre ai precedenti parametri, la struttura IORequest contiene come primo

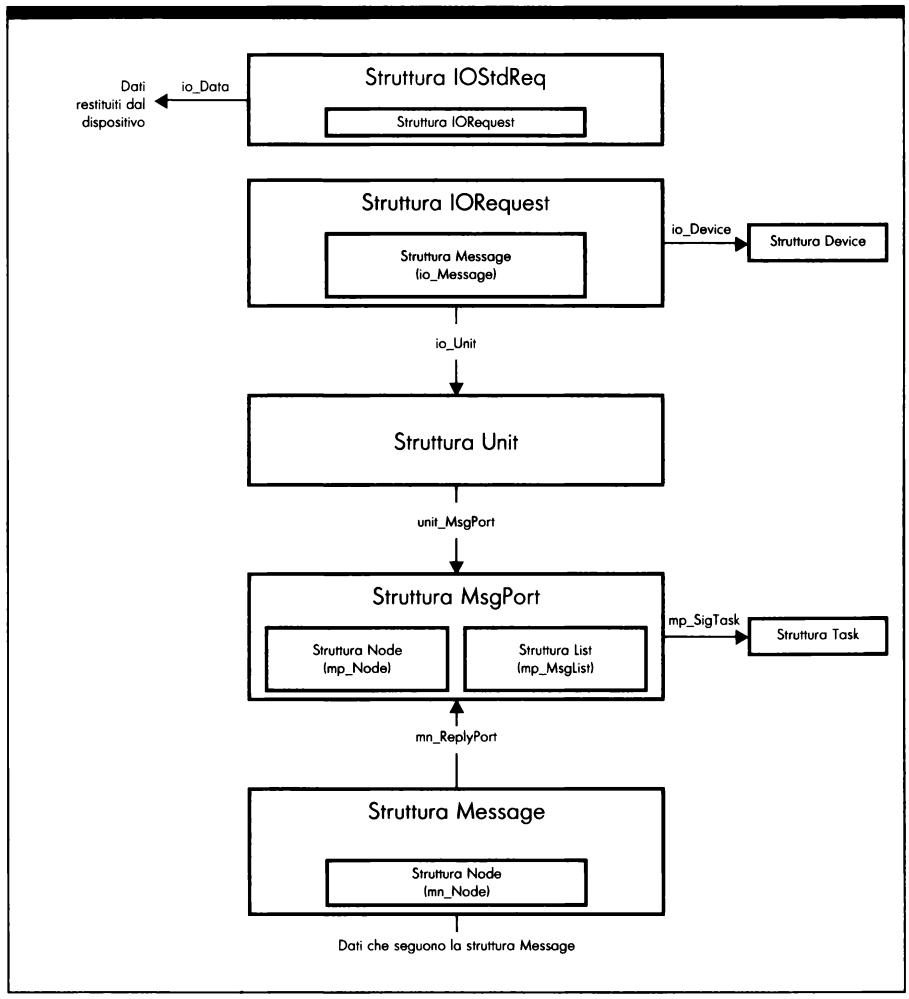


Figura 1.9: *Legami fra le diverse strutture che intervengono nelle interazioni fra task e dispositivo*

parametro una sotto-struttura Message denominata io_Message, che viene utilizzata per definire tutti i parametri della richiesta di I/O. Com'è già stato evidenziato, la struttura IORequest è in pratica un messaggio, e quindi deve iniziare con l'appropriata intestazione. La struttura Message contiene un puntatore (mn_ReplyPort) che dev'essere inizializzato in modo che punti alla reply port del task mittente o di un altro task. A questa reply port perverrà la risposta, dopo che l'unità destinataria avrà finito di elaborare la richiesta. Nel corso di questo libro presupporremo sempre che i task indichino come mittente una delle loro reply port.

Nella gestione dei dispositivi, e in particolare nelle comunicazioni fra task e dispositivo, la struttura MsgPort viene impiegata fondamentalmente in due modi. Primo, per rappresentare la request port di un'unità a cui arriva una

richiesta di I/O inviata da un task; secondo per rappresentare la reply port di un task. Nel primo caso la struttura MsgPort è allocata e inizializzata dal dispositivo e nel secondo caso dal task.

Ogni struttura MsgPort contiene una sotto-struttura Node (mp_Node) e una sotto-struttura List (mp_MsgList). La struttura Node stabilisce la doppia concatenazione con gli altri nodi della lista di sistema che elenca tutte le message port allocate come message port pubbliche (sono i task a decidere se allocarle come pubbliche). La struttura List, invece, gestisce la coda della message port alla quale pervengono i messaggi. La struttura MsgPort contiene anche un puntatore alla struttura Task relativa al task a cui essa appartiene, utile al sistema qualora il task desideri essere avvertito dell'arrivo di un messaggio. Nell'interazione con i dispositivi, in genere questo messaggio costituisce la risposta a una precedente richiesta.

La struttura Message, quella che intesta qualsiasi messaggio, contiene una sotto-struttura Node denominata mn_Node, che viene utilizzata per inserire le richieste di I/O nella coda alla request port di un'unità, oppure nella coda alla reply port di un task. Segue il puntatore mn_ReplyPort, di tipo MsgPort, per individuare la reply port a cui il messaggio dev'essere restituito, e infine il parametro mn_Length che deve specificare la lunghezza del messaggio. Si ricordi che le strutture IORequest e IOStdReq (come qualsiasi struttura di I/O tipica di un dispositivo) sono in realtà dei messaggi standard, composti da una struttura Message seguita da una serie di dati predefinita. Quindi questo parametro non viene preso in considerazione nell'interazione con i dispositivi. Illustrando le funzioni CreateExtIO e DeleteExtIO di supporto alla libreria Exec (nel prossimo capitolo), vedremo che questo parametro viene usato per altri scopi.

Strutture di I/O generali nel sistema Amiga

La programmazione dei dispositivi dell'Amiga richiede al programmatore la conoscenza di cinque strutture fondamentali: IORequest, IOStdReq, MsgPort, Message e Unit. Ognuna di queste strutture possiede un certo numero di parametri che controllano l'elaborazione delle richieste di I/O. MsgPort e Message, però, sono strutture molto più generali che non vengono impiegate solo con i dispositivi, ma in qualsiasi scambio di messaggi all'interno del sistema.

Un task definito dal programmatore deve lavorare in completa sinergia con le routine del sistema e del dispositivo per fornire e prelevare le informazioni scambiate con i dispositivi. Le strutture citate sono i veicoli di queste informazioni, ed è quindi indispensabile conoscerle a fondo.

La struttura IORequest

La struttura IORequest è la struttura standard per le richieste di I/O ai dispositivi. Si tratta di un messaggio predefinito. nel quale i parametri che

costituiscono l'informazione sono quelli attesi dai dispositivi nelle richieste di I/O. Per rendersene conto è sufficiente osservare che il primo parametro di questa struttura è una struttura Message. La struttura IORequest è definita come segue:

```
struct IORequest {
        struct Message io_Message;
        struct Device *io_Device;
        struct Unit *io_Unit;
        UWORD io_Command;
        UBYTE io_Flags;
        BYTE io_Error;
};
```

Questi sono i parametri della struttura IORequest:

- io_Message. È la sotto-struttura Message che intesta la richiesta di I/O. Questa sotto-struttura viene utilizzata dal dispositivo per conoscere il mittente a cui far pervenire la risposta alla richiesta di I/O una volta conclusa la sua elaborazione, e per accodare la richiesta nelle varie code (tramite la sotto-struttura Node).

- io_Device. È il puntatore alla struttura Device che rappresenta il dispositivo con il quale si intende comunicare. Viene automaticamente inizializzato dalle routine di sistema dell'Exec quando il dispositivo viene aperto dal task tramite la funzione OpenDevice. Si ricordi che la struttura Device è formalmente identica alla struttura Library, analizzata nel primo volume. Nella struttura Device riveste particolare importanza il parametro lib_OpenCnt, che indica il numero delle volte che il dispositivo è stato aperto e non ancora chiuso. L'accesso in scrittura al parametro lib_OpenCnt deve avvenire solo da parte del dispositivo. Per il programmatore e i suoi task non occorre sapere altro riguardo al parametro io_Device, dal momento che è il dispositivo a servirsene autonomamente: è sufficiente che per ogni richiesta di I/O venga aggiornato con il valore restituito dalla funzione OpenDevice all'apertura del dispositivo. Si tenga presente che è il parametro io_Device che indica a quale dispositivo è diretta la richiesta di I/O.

- io_Unit. È il puntatore alla struttura di tipo Unit che rappresenta la particolare unità del dispositivo con la quale s'intende comunicare. Viene automaticamente impostato dalle routine di sistema dell'Exec quando l'unità viene aperta tramite la funzione OpenDevice. Oltre al parametro unit_MsgPort che rappresenta la request port dell'unità, nella struttura Unit riveste particolare importanza il parametro unit_OpenCnt, che viene aggiornato dal dispositivo per indicare il numero delle volte che l'unità è stata aperta e non ancora chiusa. Anche per il parametro unit_OpenCnt l'accesso in scrittura deve avvenire solo da parte del dispositivo. A proposito della struttura Unit, si noti che a

differenza della struttura Device non tutti i dispositivi la impiegano; inoltre, anche quando la impiegano non sempre mantengono aggiornato il parametro unit_OpenCnt nel modo descritto. Comunque, per il programmatore e i suoi task non occorre sapere altro riguardo al parametro io_Unit, dal momento che è il dispositivo a servirsene autonomamente: è sufficiente che per ogni richiesta di I/O venga aggiornato con il valore restituito dalla funzione OpenDevice all'apertura dell'unità. Si tenga presente che è il parametro io_Unit che indica a quale unità del dispositivo è diretta la richiesta di I/O.

- io_Command. Deve sempre contenere il comando che si desidera impartire all'unità. Può trattarsi di uno dei comandi standard o di un comando specifico del particolare dispositivo.

- io_Flags. È un insieme di flag che caratterizzano ulteriormente la richiesta di I/O. I flag sono suddivisi in due campi di quattro bit ciascuno. I quattro bit di ordine più basso (dal bit 0 al bit 3) sono utilizzati dalle routine di sistema dell'Exec, mentre i quattro bit di ordine più alto (dal bit 4 al bit 7) sono a disposizione di ciascun dispositivo per le proprie necessità. I flag hanno significati diversi per ogni dispositivo (a parte il bit 0). Si ricordi che se nell'invio di un comando s'impiegano le funzioni DoIO o SendIO, questo parametro viene sempre azzerato. Quindi, se si desidera impostare uno o più flag occorre inviare la richiesta tramite BeginIO.

- io_Error. Riporta uno stato d'errore in seguito all'esecuzione di un comando da parte di un'unità. Si tratta di un codice d'errore restituito al task chiamante perché possa consultarlo e procedere di conseguenza. Gli errori di I/O si dividono in due categorie: errori standard ed errori specifici dei dispositivi.

Il flag 0 del parametro io_Flags (presente nelle strutture IORequest e IOStdReq) ha il seguente significato:

- IOF_QUICK. Impostando questo flag, quando si invia la richiesta tramite la funzione BeginIO si richiede il QuickIO. In questo caso il dispositivo, se le condizioni lo permettono, elabora la richiesta di I/O immediatamente. Se invece non può trattare la richiesta come QuickIO, essa viene accodata come una richiesta qualunque. Questo flag corrisponde al bit 0 del parametro io_Flags. Si vedano i singoli capitoli per i significati che gli altri flag del parametro io_Flags possono assumere per ogni dispositivo.

La struttura IOStdReq

La struttura IOStdReq costituisce un'estensione della struttura IORequest, cioè un messaggio predefinito più complesso della richiesta standard. Per

convincersene basta osservare che la struttura IOStdReq contiene di fatto i parametri della struttura IORequest, nello stesso ordine. La struttura IOStdReq è definita come segue:

```
struct IOStdReq {
        struct Message io_Message;
        struct Device *io_Device;
        struct Unit *io_Unit;
        UWORD io_Command;
        UBYTE io_Flags;
        BYTE io_Error;
        ULONG io_Actual;
        ULONG io_Length;
        APTR io_Data;
        ULONG io_Offset;
};
```

I primi sei parametri della struttura IOStdReq (io_Message, io_Device, io_Unit, io_Command, io_Flags e io_Error) sono gli stessi contenuti nella struttura IORequest. Gli altri sono i seguenti:

- io_Actual. Rappresenta l'effettivo numero di byte trasferiti durante l'operazione di I/O. Il suo contenuto si può ritenere valido soltanto dopo il completamento dell'I/O. Non tutti i dispositivi restituiscono un valore in questo parametro.

- io_Length. Viene in genere inizializzato dal task per indicare all'unità il numero di byte da trasferire. Un valore pari a −1 indica trasferimenti di dati di lunghezza variabile, il cui termine è indicato tramite alcune condizioni di EOF (end of file, fine del file). I caratteri di EOF, sono definiti separatamente per ciascun dispositivo. Non tutti i dispositivi richiedono espressamente un valore in questo parametro.

- io_Data. Individua in memoria un buffer definito dal task per il trasferimento dei dati fra task e dispositivo. Su questo buffer il task ha il più completo controllo.

- io_Offset. Specifica un numero di byte di offset per i dispositivi che trattano i dati strutturandoli a byte-offset, come per esempio i disk drive controllati dal dispositivo TrackDisk. Questo numero dev'essere un multiplo della dimensione del blocco relativo al particolare dispositivo (per esempio 512 byte per un dispositivo di accesso ai floppy disk).

La struttura Unit

La struttura Unit contiene i parametri con i quali vengono gestite le unità del dispositivo. Ogni unità aperta è caratterizzata dalla stessa struttura Unit.

La struttura Unit è definita come segue:

```
struct Unit {
    struct MsgPort unit_MsgPort;
    UBYTE unit_flags;
    UBYTE unit_pad;
    UWORD unit_OpenCnt;
};
```

Questi sono i parametri della struttura Unit:

- unit_MsgPort. Si tratta di una struttura MsgPort utilizzata come request port per accodare all'unità tutte le richieste di I/O provenienti dai task. Se l'unità è stata aperta nel modo di accesso condiviso, la message port può essere condivisa da più task.

- unit_flags. Contiene una serie di flag che informano sullo stato dell'unità. Si veda più avanti la loro definizione.

- unit_pad. È costituito da un byte che viene utilizzato soltanto per mantenere l'allineamento alle word nella struttura Unit, cioè per fare in modo che il numero totale di byte occupato in memoria dalla struttura sia un multiplo di due.

- unit_OpenCnt. Indica quante volte l'unità è stata aperta ma non ancora chiusa. Viene incrementato o decrementato ogni volta che un task apre o chiude l'unità.

I flag del parametro unit_flags hanno i seguenti significati:

- UNITF_ACTIVE. Se questo flag risulta impostato significa che l'unità è attiva e sta accedendo alle routine interne del dispositivo per elaborare una richiesta di I/O.

- UNITF_INTASK. Se questo flag risulta impostato significa che l'unità è associata a un particolare task. Di conseguenza, se l'unità è stata aperta dal task nel modo di accesso esclusivo, un altro task non può aprirla fino a quando il primo non l'ha chiusa.

Entrambi i flag vengono direttamente controllati dal sistema, e i task possono consultarli prima d'inviare una richiesta.

La struttura MsgPort

La struttura MsgPort contiene i parametri necessari per definire una message port. Quando un messaggio perviene a una message port, significa che è stato inserito, tramite la sua sotto-struttura Node, nella lista individuata dal

parametro mp_MsgList della struttura MsgPort. Questa è una lista doppia di nodi (ogni nodo è una struttura Node) gestita come una coda. Il fatto che venga gestita come una coda rende apparentemente inutile la struttura a doppia concatenazione (che prevede per ogni nodo un puntatore al nodo successivo e al nodo precedente), ma non bisogna dimenticare che il sistema consente anche di estrarre un messaggio prima che sia giunto alla sommità della coda, e per farlo in maniera efficiente occorre la doppia concatenazione dei nodi. La struttura MsgPort è definita come segue.

```
struct MsgPort {
    struct Node mp_Node;
    UBYTE mp_Flags;
    UBYTE mp_SigBit;
    struct Task *mp_SigTask;
    struct List mp_MsgList;
};
```

Questi sono i parametri relativi alla struttura MsgPort:

- mp_Node. È una sotto-struttura di tipo Node che viene utilizzata dal sistema per inserire la message port nella lista di sistema che raccoglie tutte le strutture MsgPort allocate in memoria come message port pubbliche. La struttura Node contiene il puntatore ln_Name, che è consigliabile utilizzare per dotare di un nome la message port, quando si desidera che sia pubblica. Una volta che il parametro ln_Name è stato definito, una serie di task possono accedere a questa message port (individuandola con il suo nome) per aggiungere o togliere richieste di I/O. Nell'interazione con i dispositivi è del tutto facoltativo rendere pubblica la message port.

- mp_Flags. Contiene una serie di flag per la struttura MsgPort. Questi flag indicano al sistema Exec cosa deve accadere quando la message port riceve un messaggio nella propria coda. Si veda più avanti il significato dei singoli flag.

- mp_SigBit. È il numero del bit di segnale che viene attivato quando arriva un messaggio nella coda alla message port (purché risulti impostato il flag PA_SIGNAL del parametro mp_Flags). Il bit di segnale che viene in questo caso impostato si trova nel parametro tc_SigRecvd presente nella struttura Task relativa al task individuato dal parametro mp_SigTask della struttura MsgPort. Se il task è in attesa di questo segnale tramite la funzione Wait, riottiene il controllo nel momento in cui la message port riceve un nuovo messaggio. Se con questa stessa funzione il task è in attesa di diversi messaggi, quando riottiene il controllo può accedere al parametro tc_SigRecvd per rilevare in quale message port è giunto il messaggio. Si ricordi che quando un task chiama la funzione Wait non riceve più cicli della CPU fino a quando non perviene un segnale. Ogni message port può indicare un solo bit di

segnale nel parametro mp_SigBit, e quindi può avvertire solo un task alla volta. Si dice quindi che la message port appartiene a uno e un solo task.

- mp_SigTask. È un puntatore che individua la struttura Task relativa al task che dev'essere avvertito (sempre con un segnale) dell'arrivo di un messaggio nella coda alla message port. Il segnale viene inviato solo se nel parametro mp_Flags risulta impostato il flag PA_SIGNAL. Se il parametro mp_SigTask è stato inizializzato, si dice che la message port appartiene al task.

- mp_MsgList. È la sotto-struttura List che mantiene l'elenco di tutti i messaggi in arrivo nella message port. Quest'elenco viene gestito dall'Exec in modo FIFO (First In, First Out), e rappresenta quindi una coda. I messaggi vengono accodati tramite le sotto-strutture Node presenti nelle strutture Message che li intestano.

I flag del parametro mp_Flags presente nella struttura MsgPort hanno i seguenti significati:

- PA_SIGNAL. Se all'arrivo di un messaggio nella coda alla message port questo flag risulta impostato, il sistema invia un segnale al task indicato dal puntatore mp_SigTask.

- PA_SOFTINT. Se all'arrivo di un messaggio nella coda alla message port questo flag risulta impostato, il sistema genera un interrupt software.

- PA_IGNORE. Se all'arrivo di un messaggio nella coda alla message port questo flag risulta impostato, il messaggio viene ignorato. Questo significa che nessun segnale viene inviato dal sistema al task.

La struttura Message

La struttura Message costituisce l'intestazione dei messaggi. Nell'organizzazione di un messaggio, i dati veri e propri vengono subito dopo questa struttura. Le strutture IORequest e IOStdReq sono due esempi di messaggi predefiniti, nei quali alla struttura Message seguono i parametri che costituiscono la richiesta di I/O, cioè il messaggio vero e proprio. La struttura Message è definita come segue:

```
struct Message {
        struct Node mn_Node;
        struct MsgPort *mn_ReplyPort;
        UWORD mn_Length;
};
```

Questi sono i parametri della struttura Message:

- mn_Node. È una sotto-struttura Node che consente di accodare il messaggio nelle code alle message port. La struttura Node del messaggio crea la doppia concatenazione con le strutture Node degli altri messaggi, andando ad aggiungersi alle liste che l'Exec gestisce come code.

- mn_ReplyPort. È un puntatore che individua la struttura MsgPort della reply port alla quale il messaggio viene restituito una volta che il destinatario l'ha elaborato. È interessante notare che qualsiasi messaggio deve sempre ritornare al mittente indicato nella sua intestazione per due fondamentali ragioni. Prima di tutto perché solo quando il messaggio arriva alla sua reply port, il task ha la certezza che sia arrivato a destinazione e sia stato elaborato; solo in quel momento, quindi, può riutilizzarne la struttura per nuovi messaggi. In secondo luogo, la risposta al messaggio (che di fatto è il messaggio stesso) può essere stata modificata per restituire al mittente alcune informazioni, ovvero un nuovo messaggio. Nell'interazione task-dispositivo, il destinatario del messaggio è un dispositivo, il quale normalmente lo elabora e lo modifica, restituendo qualche informazione al task che l'ha inviato. Si ricordi che quando un messaggio viene inviato non dev'essere più alterato dal task fino a quando non viene restituito. Parallelamente, il dispositivo deve leggere, elaborare e modificare il messaggio *prima* di restituirlo: subito dopo l'invio deve considerarlo inaccessibile.

- mn_Length. Contiene il numero di byte presenti nel messaggio (64K è la lunghezza massima). Non viene utilizzato nelle comunicazioni tra task e dispositivo in quanto le richieste di I/O sono predefinite, e i dispositivi che le ricevono conoscono perfettamente le strutture dei messaggi. Nel capitolo 2 vedremo invece come questo parametro viene impiegato dalle funzioni CreateExtIO e DeleteExtIO di supporto alla libreria Exec.

file INCLUDE e le strutture relative ai dispositivi

La Tavola 1.2 (nella pagina successiva) presenta un sommario delle strutture relative ai dispositivi, e dei file INCLUDE che le definiscono. Sebbene alcuni dispositivi prevedano strutture di I/O non standard per le richieste (diverse cioè da IORequest e IOStdReq), si noterà nel corso dei prossimi capitoli che tutte queste strutture dedicate contengono sempre come primo elemento la struttura IORequest (alcune contengono la struttura IOStdReq, che però non è altro che un'estensione della struttura IORequest).

Quattro dispositivi (Console, Gameport, Input e Keyboard) richiedono ai task di specificare le proprie richieste di I/O tramite la struttura estesa standard

Tavola 1.2: *Strutture dei dispositivi e file INCLUDE nei quali sono definite*

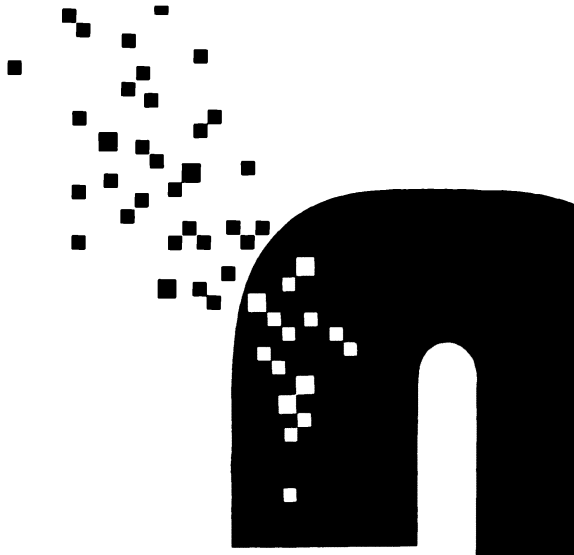
Dispositivo	Nome della struttura per la richiesta di I/O	Nome della sottostruttura appartenente alla struttura di I/O	Prima struttura ausiliaria	Seconda struttura ausiliaria	Terza struttura ausiliaria	Quarta struttura ausiliaria	File INCLUDE
Audio	IOAudio	IORequest	AudChannel	–	–	–	audio.h
Clipboard	IOClipReq	IOStdReq	Clipboard-UnitPartial	SatisfyMsg	–	–	clipboard.h
Console	IOStdReq	–	ConUnit	–	–	–	console.h conunit.h
Gameport	IOStdReq	–	GamePortTrigger	–	–	–	gameport.h
Input	IOStdReq	–	InputEvent	–	–	–	input.h inputevent.h
Keyboard	IOStdReq	–	KeyMapNode	KeyMap	KeyMap-Resource	–	keyboard.h keymap.h
Narrator	narrator_rb mouth_rb	IOStdReq	–	–	–	–	narrator.h
Parallel	IOExtPar	IOStdReq	IOPArray	–	–	–	parallel.h
Printer	IOPrtCmdReq IODRPReq	IOStdReq	PrinterData	PrinterSegment	PrinterEx-tendedData	DeviceData	prtbase.h printer.h
Serial	IOExtSer	IOStdReq	IOTArray	–	–	–	serial.h
Timer	timerequest	IORequest	timeval	–	–	–	timer.h
TrackDisk	IOExtTD	IOStdReq	TDU_Public-Unit	–	–	–	trackdisk.h

IOStdReq. Nei loro file INCLUDE non appaiono quindi definizioni di strutture di I/O, dal momento che la struttura IOStdReq e la struttura IORequest sono definite nel file INCLUDE exec/io.h.

Gli altri otto dispositivi prevedono invece una o più strutture di I/O non standard. Il dispositivo Printer riconosce per le proprie richieste due strutture di I/O: IOPrtCmdReq per le richieste di I/O generiche, e IODRPReq per le richieste di I/O relative alla stampa delle bitmap. Queste strutture, insieme con quelle previste dai dispositivi Audio e Timer, contengono la struttura IORequest come primo elemento. I dispositivi Clipboard, Narrator, Parallel, Serial e TrackDisk utilizzano invece come primo elemento della loro struttura non-standard la struttura IOStdReq.

In aggiunta a queste strutture, i dispositivi ne utilizzano spesso altre come ausilio nella gestione. I dispositivi Audio, Parallel, Serial, Timer, TrackDisk e Gameport possiedono ognuno una struttura ausiliaria. Clipboard ne possiede due, il dispositivo Keyboard tre e il dispositivo Printer quattro.

Ogni dispositivo possiede almeno un file INCLUDE che definisce strutture e dati che i task devono impiegare per comunicare correttamente con le sue routine interne. I dispositivi Console, Input, Keyboard e Printer possiedono due file INCLUDE.



2

La gestione dei dispositivi

Introduzione

Questo capitolo analizza alcuni concetti generali di fondamentale importanza per la gestione e per la programmazione dei dispositivi di I/O dell'Amiga. Questi dispositivi sono stati pensati in modo tale che un programmatore possa trarre il massimo vantaggio dalle loro routine interne, e sono molto diversi da quelli adottati nella maggior parte dei computer.

Nella prima parte del capitolo viene illustrata la gestione dei dispositivi in linguaggio C, e vengono descritti i più comuni task di gestione nonché l'ordine in cui devono essere eseguiti. Le sezioni che seguono prendono in esame le funzioni AbortIO, BeginIO, DoIO, SendIO, WaitIO, CheckIO, AddDevice e RemDevice: la base essenziale su cui costruire task di gestione dei dispositivi nei propri programmi.

Il capitolo si conclude con l'analisi delle nove funzioni di supporto alla libreria Exec. Si tratta di nove funzioni contenute nella libreria di tipo linked amiga.lib del sistema Lattice (c.lib nel sistema Manx), che sono espressamente dedicate alla gestione dei dispositivi e dei task. Non appartengono al ROM Kernel, e vengono incluse nei codici dei task durante la fase di link. Ognuna di queste funzioni è un insieme d'istruzioni il cui scopo è semplificare la gestione delle interazioni task-task e task-dispositivo. Il programmatore, anziché allocare le necessarie strutture, inizializzarle, chiamare direttamente le funzioni dell'Exec per gestire gli I/O nei propri task ed effettuare i necessari controlli, può delegare la maggior parte del lavoro a queste nove funzioni. Il fatto che siano contenute in una libreria di tipo linked permette un più facile accesso e consente di creare sorgenti più corti di quelli che si otterrebbero utilizzando altri metodi (l'appendice descrive con precisione otto delle nove funzioni di supporto alla libreria Exec).

Procedure di programmazione generali

La Figura 2.1 (nella pagina successiva) descrive la sequenza generale dei passi da seguire nella programmazione dei dispositivi. Per "programmazione dei dispositivi" si intende la creazione dei codici e delle strutture necessarie per le comunicazioni con i dispositivi e soprattutto per l'invio di comandi. Per questa ragione, nella sequenza si prevede anche la creazione di un apposito task di gestione del dispositivo oltre al task principale. Sebbene non sia assolutamente vincolante, questa scelta vuole mettere in luce i vantaggi che si ottengono sfruttando le capacità multitasking per rendere modulari i propri programmi.

La sequenza che stiamo per descrivere prevede l'apertura di un'unità del generico dispositivo tramite la funzione OpenDevice, l'invio di una serie di comandi all'unità tramite le funzioni BeginIO, DoIO o SendIO, e infine la chiusura dell'unità tramite la funzione CloseDevice. Viene presentato il caso di

un solo task, una sola unità, una sola reply port del task e alcune strutture di I/O, ma lo stesso schema si applica anche in casi più complessi, come si vedrà meglio in seguito.

Ecco quali sono i passi da seguire nella programmazione.

1. *Creare un task che si dedichi alla gestione del dispositivo*. Quest'operazione iniziale può essere svolta tramite la funzione CreateTask (funzione di supporto alla libreria Exec) all'interno di un task principale (tutti i dispositivi con i quali il sistema interagisce in modo autonomo sono gestiti da una serie di task appositamente creati da altri task di sistema;

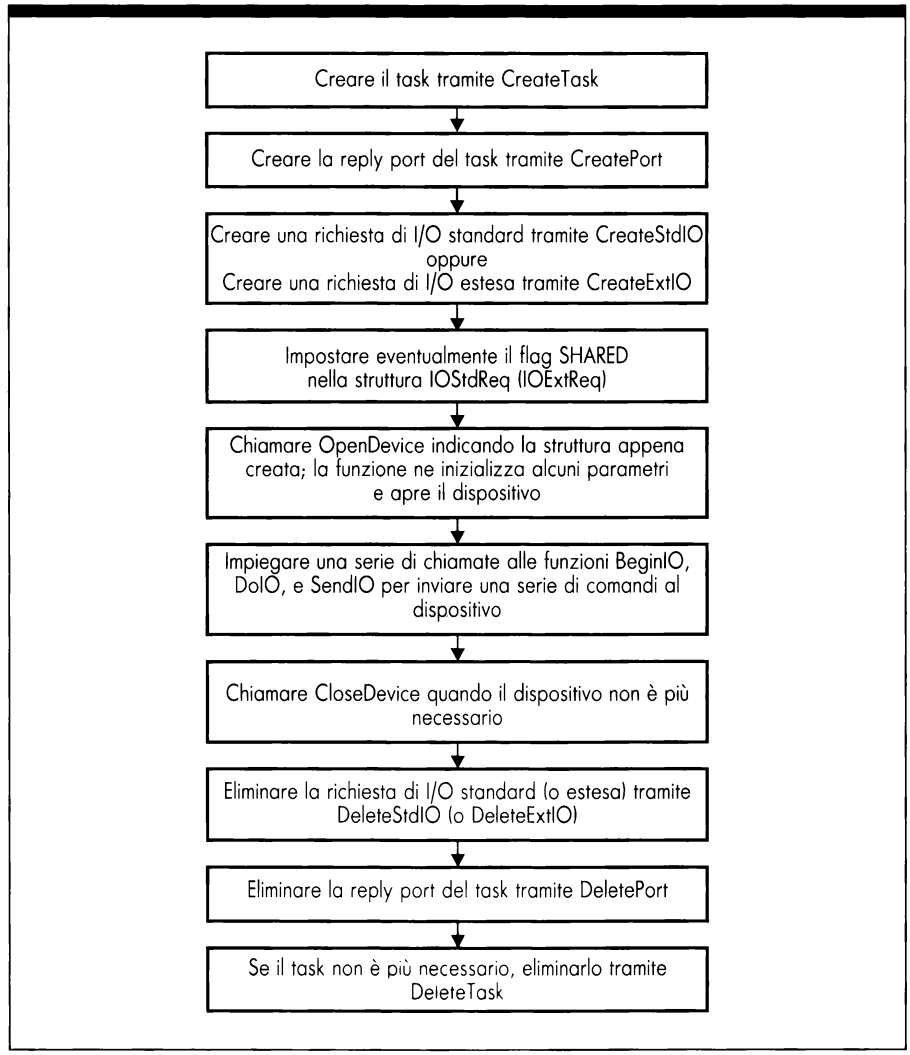


Figura 2.1: *Gestione di un dispositivo attraverso un task*

il task che effettua il boot del sistema è quello principale). Questa funzione di supporto alloca in memoria una struttura di tipo Task e la quantità di byte specificata negli argomenti della funzione per lo user stack del task. Provvede poi ad aggiornare i parametri della struttura Task a seconda dei valori indicati negli argomenti della chiamata (nome del task, priorità, punto d'ingresso, dimensione dello stack). Se il primo argomento della chiamata a CreateTask è un indirizzo non nullo che individua in memoria una stringa di testo, la funzione lo memorizza nel puntatore ln_Name della sotto-struttura Node presente nella relativa struttura Task. Questa stringa rappresenta il nome tramite il quale è possibile individuare il task nella lista di sistema TaskList. Infine, CreateTask provvede ad aggiungere il nuovo task nella lista di sistema TaskList tramite la funzione AddTask. A questo punto tutti gli altri task possono ottenere, tramite la funzione FindTask della libreria Exec, l'indirizzo della sua struttura Task. Si noti che senza la funzione di supporto CreateTask, i programmatori dovrebbero inserire all'interno dei sorgenti tutti i codici necessari per eseguire le operazioni appena illustrate. CreateTask limita il lavoro alla semplice chiamata di una funzione.

Nella creazione di task dall'interno di altri task c'è un aspetto di cui occorre assolutamente tenere conto: se il task che si crea non è subordinato al funzionamento del task che l'ha creato (cioè se deve continuare a funzionare anche quando il task che l'ha generato viene rimosso), dev'essere allocato in un'area di memoria esterna a quella del primo task, altrimenti rimuovendo il primo task verrebbe anch'esso rimosso. Quindi il suo codice, la relativa struttura Task, lo stack e tutti gli altri dati che descrivono il secondo task devono essere allocati dinamicamente.

2. *Creare una message port da associare al task di gestione del dispositivo utilizzando la funzione di supporto CreatePort*. Questa message port rappresenta la reply port del task alla quale i dispositivi devono far pervenire le risposte alle richieste di I/O. La funzione CreatePort dev'essere chiamata all'interno del task di gestione del dispositivo. Anche l'unità del dispositivo possiede una message port (quella che chiamiamo request port) per ricevere le richieste di I/O a essa indirizzate. Questa message port è controllata dalla sotto-struttura MsgPort contenuta nella struttura Unit dell'unità: crearla non è compito dei task di gestione del dispositivo.

   Un task di gestione dovrebbe possedere una reply port per ciascuna possibile categoria di richieste di I/O (questo significa disporre di una message port separata per ogni unità del dispositivo aperta).

3. *Creare una struttura di I/O utilizzando le funzioni di supporto CreateStdIO oppure CreateExtIO*. Questa struttura verrà poi impiegata per inviare le richieste di I/O all'unità aperta. Si deve utilizzare la prima funzione solo se il dispositivo prevede che le richieste di I/O a esso indirizzate abbiano il formato della struttura standard IOStdReq. Se

invece il dispositivo richiede una struttura di I/O non standard, si deve impiegare la funzione CreateExtIO. Ovviamente, queste due funzioni di supporto alla libreria Exec si preoccupano solo di allocare la memoria occupata dal messaggio e d'inizializzare i parametri dell'intestazione, mentre è compito del task fornire il contenuto vero e proprio del messaggio, in conformità con quanto si aspetta il dispositivo. La struttura di I/O è ora allocata in memoria, non è accodata in nessuna coda, e si dice che "appartiene" al task.

4. *Inizializzare gli appropriati flag contenuti nella struttura di I/O (se il dispositivo ne prevede l'impiego).* Se si deve aprire il dispositivo TrackDisk, occorre anche indicare i flag previsti dalla funzione OpenDevice nell'argomento flags. Inoltre, occorre decidere se aprire l'unità nel modo di accesso esclusivo o nel modo di accesso condiviso (con i dispositivi che offrono questa scelta), ed eventualmente inizializzare altri parametri; consultare i capitoli relativi ai singoli dispositivi per sapere quali sono questi parametri.

5. *Chiamare la funzione OpenDevice per aprire l'unità del dispositivo.* Questa funzione provvede a caricare il dispositivo in memoria e inizializzarlo qualora si tratti di un dispositivo residente su disco e non si trovi già in memoria (il sistema, per sapere se il dispositivo è disponibile, cioè in memoria, lo cerca all'interno della lista di sistema DeviceList; se non lo trova, ne deduce che si tratta di un dispositivo residente su disco non ancora caricato in memoria). Poi OpenDevice incrementa automaticamente il parametro lib_OpenCnt della struttura Device per segnalare l'apertura del dispositivo. Per segnalare anche l'apertura dell'unità viene inoltre incrementato automaticamente il parametro unit_OpenCnt della sua struttura Unit. Dopo queste operazioni, la funzione OpenDevice inizializza in modo opportuno alcuni parametri della struttura di I/O e aggiorna i parametri io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit.

6. *Effettuare le opportune chiamate alle funzioni BeginIO, DoIO e SendIO per inviare i comandi all'unità.* Occorre prima di tutto decidere quali sono le esigenze che il dispositivo deve soddisfare. Quindi si decide se il task deve addormentarsi in attesa di ricevere i dati richiesti (I/O sincrono), oppure se può richiederli e, senza averli ancora ricevuti, procedere allo svolgimento di altre mansioni (I/O asincrono). Infine si procede all'invio dei comandi, assicurandosi che nelle relative strutture di I/O i parametri io_Device e io_Unit contengano i valori restituiti dalla funzione OpenDevice all'apertura del dispositivo.

7. *Chiudere l'unità del dispositivo quando si è sicuri che il task non ne ha più bisogno.* In generale occorre chiamare la funzione CloseDevice utilizzando la stessa struttura di I/O impiegata nella chiamata a OpenDevice. Questa operazione produce la decrementazione del

parametro unit_OpenCnt della struttura Unit e del parametro lib_OpenCnt della struttura Device. Se il task non ha aperto nel frattempo altre unità, l'accesso al dispositivo si ritiene chiuso. Si noti che chiamando la funzione CloseDevice si può anche provocare l'azzeramento del parametro lib_OpenCnt della struttura Device del dispositivo. Anche in questo caso, però, il dispositivo continua a restare disponibile: la sua struttura Device non viene rimossa dalla lista DeviceList, e se si trova in memoria RAM non viene disallocato. Come vedremo, l'unico modo che hanno i task per liberare la memoria occupata da un dispositivo disponibile (se si tratta di un dispositivo residente su disco) è chiamare la funzione RemDevice.

8. *Liberare la memoria occupata dalla struttura di I/O*. Si utilizza la funzione DeleteStdIO se si era fatto ricorso a CreateStdIO, o la funzione DeleteExtIO se si era fatto ricorso a CreateExtIO. Queste funzioni non liberano però la memoria occupata dalla reply port del task, che può essere usata per moltissimi altri scopi.

9. *Eliminare la struttura MsgPort che costituiva la reply port del task*. Si chiama la funzione DeletePort, che libera la memoria occupata dalla struttura MsgPort. Tuttavia, se si desidera utilizzare o si sta già utilizzando la message port per altri scopi, ovviamente si salta questo passo.

10. *Eliminare il task per la gestione del dispositivo*. Si chiama dal task principale la funzione DeleteTask per liberare la memoria occupata dalla struttura Task e dallo stesso task di gestione del dispositivo (anche questa operazione è da ritenersi opzionale).

Elaborazione delle richieste di I/O asincrono

La Figura 2.2 (nella pagina successiva) mostra il comportamento del sistema durante l'elaborazione di richieste di I/O asincrono. L'I/O asincrono viene utilizzato quando un task desidera inviare diverse richieste di I/O una dopo l'altra, ma non ha assoluto bisogno dei dati richiesti per proseguire il suo corso. Le funzioni che controllano il flusso delle richieste di I/O asincrono sono cinque: SendIO, CheckIO, WaitIO, AbortIO (che appartengono alla libreria Exec), e BeginIO (definita all'interno di ogni dispositivo). Oltre a queste, anche le funzioni GetMsg e Remove (della libreria Exec) possiedono un ruolo rilevante nella sequenza delle azioni necessarie per le richieste di I/O asincrono.

I sei rettangoli nella figura rappresentano una sequenza di azioni, una parte delle quali è sotto il controllo del task definito dal programmatore. La maggior parte però è determinata e controllata dalle azioni coordinate del sistema e delle routine interne del dispositivo. Il QuickIO non è previsto nell'I/O asincrono e quindi non viene preso in considerazione. In questo paragrafo, dunque, si suppone che se il task invia il comando tramite la funzione BeginIO con il flag IOF_QUICK impostato, il dispositivo non accordi il QuickIO, e quindi

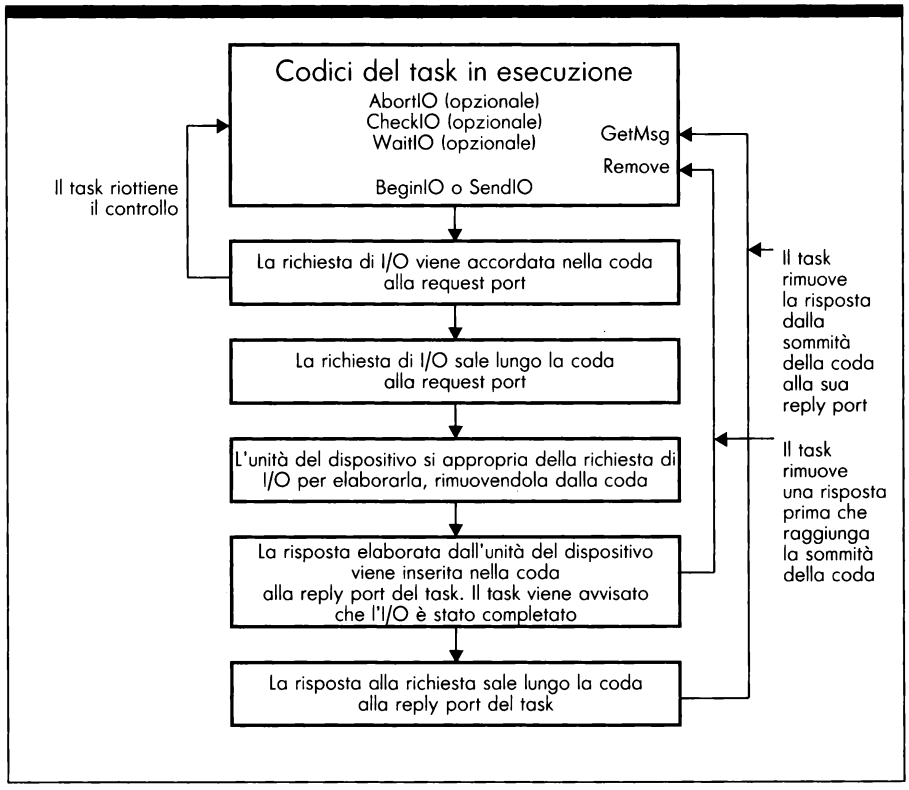


Figura 2.2: *Evoluzione di una richiesta di I/O asincrono*

che la funzione BeginIO si comporti in modo asincrono. Questo problema è superfluo nel caso di SendIO, dal momento che questa funzione azzera sempre il parametro io_Flags, e quindi anche il flag IOF_QUICK.

Ecco la sequenza di azioni seguita nel caso dell'I/O asincrono.

1. Il task invia una richiesta di I/O asincrono utilizzando la funzione BeginIO oppure SendIO (si noti che l'unità del dispositivo è già stata aperta tramite OpenDevice). Trattandosi di un invio asincrono, il task non viene addormentato e può procedere con le sue mansioni.

2. La richiesta di I/O viene inserita dal dispositivo nella coda alla request port dell'unità (si utilizzano le strutture Device, Unit, MsgPort e Message). La coda alla request port dell'unità è di tipo FIFO (First In, First Out), e quindi questa richiesta di I/O viene collocata nella parte più bassa, in attesa di giungere alla sommità; convenzionalmente viene detto che "appartiene alle routine interne del dispositivo", nel senso che la richiesta non deve più essere alterata dal task che l'ha inoltrata fino a quando non la riceve di ritorno alla sua reply port).

   Trattandosi di una richiesta asincrona, ora il task può proseguire

nello svolgimento di altre mansioni. Se però chiama la funzione WaitIO, il task entra in stato d'attesa perdendo il controllo della CPU e vi rimane fino a quando il dispositivo non restituisce la richiesta. Si tenga presente che la funzione WaitIO addormenta il task in attesa che giunga la risposta a una particolare richiesta, e quindi è da considerarsi una funzione rischiosa, in quanto se per qualche ragione la risposta alla richiesta non viene restituita alla reply port, il task non viene più riattivato. Per attendere l'arrivo di messaggi nelle code alle message port è più sicuro impiegare le funzioni Wait e WaitPort. La prima attende i segnali provenienti da una o più porte. La seconda attende finché non arriva un messaggio qualsiasi a una particolare message port. Evidentemente, è più improbabile che queste due funzioni non restituiscano più il controllo al task.

Un'altra possibilità (senza addormentare il task), è quella di eseguire controlli periodici con la funzione CheckIO, la quale rileva se una particolare struttura di I/O (più in generale un particolare messaggio) è stata restituita al mittente.

3. La richiesta di I/O ha finalmente terminato il suo tragitto arrivando alla sommità della coda alla request port, e può quindi essere elaborata dalle routine interne del dispositivo. Il tempo impiegato dalla richiesta per compiere questa "ascesa" dipende dalla lunghezza della coda e dall'attività generale del sistema. La quantità di compiti di cui il sistema si sta prendendo cura determina infatti la quantità di tempo che la CPU può dedicare alle routine interne del dispositivo in questione, e quindi la celerità con cui questo rimuove la richiesta di I/O dalla coda alla request port e la elabora.

4. Le routine interne del dispositivo rimuovono la richiesta di I/O dalla sommità della coda. Il dispositivo utilizza una funzione analoga a GetMsg per prelevare la richiesta, quindi accede ai parametri della struttura per decifrare il comando del task mittente.

5. Il dispositivo restituisce la risposta collocandone la struttura di I/O nella parte bassa della coda alla reply port del task. Se si è verificato qualche imprevisto durante l'elaborazione, il parametro io_Error della struttura di I/O restituita dal dispositivo contiene il codice del relativo errore altrimenti contiene il valore 0. Per conoscere il mittente, il dispositivo accede al parametro mn_ReplyPort della struttura Message contenuta nella struttura di I/O (il dispositivo ricorre poi alla funzione ReplyMsg per inviare la risposta alla reply port del task). Inoltre, se è stato impostato un meccanismo di segnalazione tra il task e la sua reply port, quando giunge il messaggio nella coda alla reply port il task viene subito avvertito.

6. Ora dobbiamo analizzare diversi casi, a seconda del modo che il task ha scelto per attendere la risposta.

Se dopo l'invio della richiesta di I/O, eventualmente dopo aver

compiuto qualche altra operazione, il task ha chiamato la funzione WaitIO indicando come parametro l'indirizzo della struttura di I/O (ed è entrato quindi in stato d'attesa), quando (e se) riottiene il controllo significa che la risposta è arrivata alla sua reply port ed è stata rimossa da WaitIO. A questo punto il task ha la certezza di poter riaccedere alla struttura di I/O che costituisce la risposta ed esaminarla, in quanto non è più di proprietà del dispositivo. Il task può quindi procedere ad elaborare il messaggio "saltando" gli altri messaggi in coda alla sua reply port.

Una diversa situazione si presenta quando il task ha scelto di chiamare periodicamente la funzione CheckIO, indicando come parametro l'indirizzo della struttura della richiesta di I/O per la quale attende risposta. A ogni chiamata, CheckIO restituisce il valore 0 se la risposta non è ancora arrivata, altrimenti l'indirizzo della struttura di I/O (cosa perfettamente inutile, in quanto il task, avendolo dovuto indicare come parametro della funzione, lo conosce perfettamente). La grande differenza rispetto a WaitIO è che CheckIO non addormenta il task, e inoltre non provvede a rimuovere la richiesta dalla coda alla reply port. Quando CheckIO restituisce un indirizzo diverso da zero, il task può procedere all'analisi diretta della risposta e alla sua rimozione dalla coda tramite la funzione Remove (si noti che rimuovere una richiesta da una coda non significa disallocarla, ma solo estrarre dalla lista a doppia concatenazione la sua sotto-struttura Node). Se il task non la rimuove, la risposta continua la sua ascesa nella coda.

La terza situazione che si può presentare è quando il task, eventualmente dopo aver compiuto qualche altra operazione, chiama la funzione Wait indicando come parametri uno o più bit di segnale; questa funzione lo addormenta fino a quando in una delle code alle message port relative a questi bit di segnale non giunge un qualsiasi messaggio. Ovviamente, per usare la funzione Wait il task deve aver allocato gli opportuni bit di segnale nelle sue reply port. Quando riottiene il controllo, il task non può sapere se il messaggio ricevuto in una delle sue reply port è la risposta alla sua richiesta di I/O. La prima operazione che deve compiere è quindi controllare in quale reply port è giunto il messaggio, analizzando il numero del segnale. Successivamente, può stabilire tramite la funzione CheckIO se il messaggio pervenuto è la risposta che sta aspettando. Se il risultato è positivo, procede come nel caso precedente.

La quarta e ultima situazione che si può presentare è quando il task, eventualmente dopo aver compiuto qualche altra operazione, chiama la funzione WaitPort indicando come parametro la struttura MsgPort della sua reply port. In questo caso resta addormentato per tutto il tempo che la coda a quella reply port risulta vuota. Quando riottiene il controllo significa che alla reply port è giunto un messaggio, ma il task non ha la certezza che si tratti della risposta alla richiesta di I/O che ha precedentemente inoltrato. La funzione WaitPort restituisce l'indirizzo del messaggio ma non lo rimuove. Il task confronta l'indirizzo ottenuto con quello della struttura di I/O che aveva impiegato per la richiesta,

oppure procede con la funzione CheckIO come nei casi precedenti.

Finora abbiamo sempre supposto che il task desiderasse accedere subito alla risposta senza elaborare tutti i messaggi che la precedevano nella coda alla reply port, e che la rimuovesse subito tramite la funzione Remove. La figura mostra questa situazione con la linea che partendo dal quinto rettangolo raggiunge il primo. Però dev'essere chiaro che il task può anche decidere di analizzare la risposta solo quando è giunta alla sommità della coda, dopo la necessaria sequenza di chiamate alla funzione GetMsg.

7. Se il task, quando riottiene il controllo da una delle funzioni precedentemente elencate (esclusa WaitIO), non provvede a rimuovere la risposta dalla coda alla reply port utilizzando la funzione Remove, può vagliare sequenzialmente i messaggi presenti nella coda e identificare fra questi la risposta attesa. I messaggi avanzano di posizione nella coda alla reply port a mano a mano che il task chiama GetMsg, dal momento che la stessa funzione provvede anche a rimuoverli. Questa procedura è mostrata nella figura dalla linea che parte dal sesto rettangolo e raggiunge il primo.

Quest'ultimo stadio completa il cammino di una richiesta di I/O asincrono attraverso il sistema, dal task mittente al dispositivo, e dal dispositivo di nuovo al task. La richiesta inviata all'unità tramite BeginIO o SendIO passa attraverso le due code e infine ritorna al task che l'ha originata, il quale può quindi accedere ai dati elaborati e se necessario utilizzare ancora la stessa struttura di I/O.

Per quanto riguarda l'I/O asincrono, occorre aprire una parentesi a proposito dei comandi immediati, cioè quei comandi che il dispositivo esegue subito, senza accodarli a nessuna request port: CMD_START, CMD_STOP, CMD_RESET... Per inviarli, i task possono servirsi indistintamente delle funzioni SendIO, BeginIO o DoIO. Questi comandi hanno precedenza assoluta, anche perché non devono intervenire solo sull'unità, ma anche sulla richiesta in corso di elaborazione. È per questo che non vengono mai accodati, e i task non hanno la facoltà di cambiare tale caratteristica. Possono però intervenire sul modo in cui il dispositivo restituisce la richiesta una volta che il comando immediato è stato eseguito. Impostando il flag IOF_QUICK nel parametro io_Flags della richiesta, il dispositivo assume nella restituzione un comportamento sincrono. Se invece il flag IOF_QUICK risulta azzerato, il dispositivo restituisce la richiesta di I/O alla reply port del task utilizzando una funzione analoga a ReplyMsg. Questo comportamento si verifica quando il task invia il comando immediato tramite la funzione SendIO, o BeginIO con il flag IOF_QUICK azzerato.

Elaborazione delle richieste di I/O sincrono

La Figura 2.3 (nella pagina successiva) mostra il modo in cui il sistema si comporta durante l'elaborazione di una richiesta di I/O sincrono. Generalmente

un task utilizza le richieste di I/O sincrono quando ha necessità d'inviare una richiesta di I/O per volta e di ricevere dal dispositivo i dati desiderati prima di proseguire nello svolgimento di altre mansioni. In questo caso si possono presentare tre diverse situazioni, a seconda che il task impieghi DoIO o BeginIO (con il flag IOF_QUICK impostato), o infine che il comando sia di tipo immediato.

Esistono alcune importanti differenze tra l'elaborazione sincrona e quella asincrona, com'è evidenziato dalle due figure. Per quanto riguarda l'I/O sincrono questi sono i punti chiave:

- non vengono utilizzate le funzioni SendIO, CheckIO, WaitIO e GetMsg; queste funzioni servono solo per le richieste di I/O asincrono.

- Quando il task chiama la funzione BeginIO (richiedendo il QuickIO e supponendo che sia accordato) oppure DoIO, perde il controllo della CPU e rimane in attesa fino a quando il dispositivo non conclude l'elaborazione della richiesta. Quando il task riottiene il controllo, si presentano due casi. Primo: se ha impiegato BeginIO deve verificare lo stato del flag IOF_QUICK (se risulta impostato significa che il QuickIO è stato accolto, cioè la richiesta è stata elaborata subito e quindi in modo sincrono, e che la risposta non è stata accodata alla sua reply port). La funzione BeginIO si comporta in modo sincrono solo quando il task ha richiesto il QuickIO e il dispositivo l'ha accordato; altrimenti, la richiesta

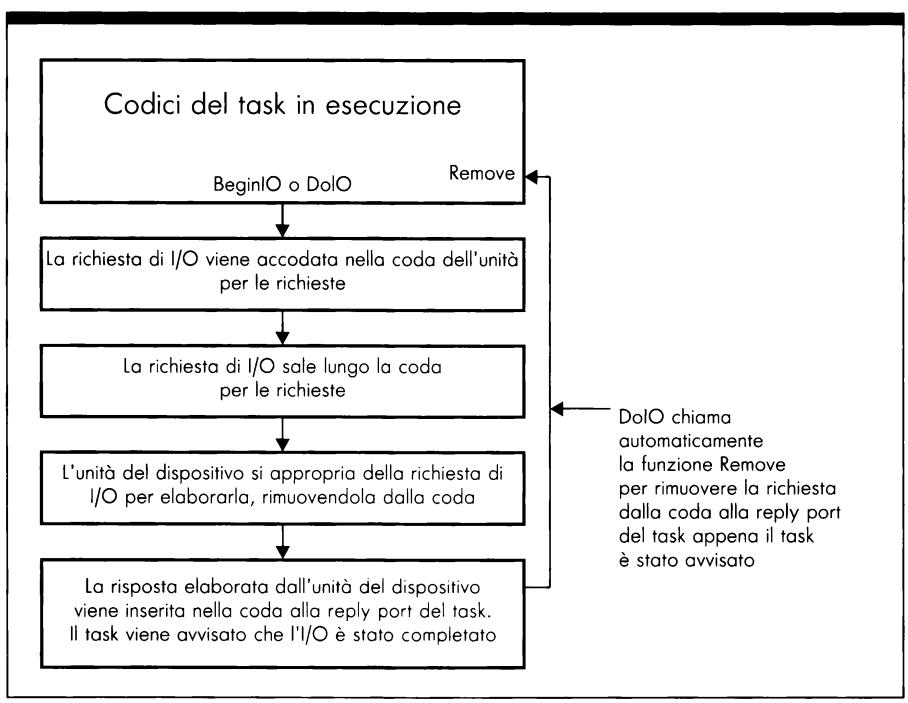


Figura 2.3: *Evoluzione di una richiesta di I/O sincrono*

viene trattata in modo asincrono e si torna a quanto illustrato nella precedente sezione. Secondo: il task ha impiegato DoIO (che richiede il QuickIO automaticamente) ma comunque, sia che l'accesso veloce venga accordato sia che non venga accordato, riottiene il controllo solo quando il dispositivo ha concluso l'elaborazione. La funzione DoIO, agli occhi del task, si comporta sempre in modo sincrono: in entrambi i casi, quando il task riottiene il controllo può accedere ai dati restituiti dal dispositivo (si noti che DoIO prima di restituire il controllo provvede automaticamente a rimuovere la richiesta dalla reply port del task, qualora il QuickIO non fosse stato accordato). Durante questo intervallo di tempo, mentre il task è in attesa, altri task possono appropriarsi della CPU. Questo è il motivo principale per cui si utilizza l'I/O sincrono.

- Come abbiamo già detto, se con la funzione BeginIO il QuickIO non viene accordato, il processo diventa asincrono. Se invece il QuickIO non viene accordato a una richiesta inviata con DoIO, il task non si accorge di niente, perché DoIO attende per lui la risposta alla reply port, la rimuove quando arriva, e solo allora restituisce il controllo. Quindi, anche se in effetti la richiesta viene trattata dal dispositivo in modo asincrono, DoIO camuffa il tutto in modo che al task l'accesso sembri sempre sincrono. Quando illustreremo nei dettagli il funzionamento di DoIO, questo aspetto risulterà più evidente.

- Trattando l'I/O asincrono abbiamo visto che azzerando il flag IOF_QUICK i comandi immediati vengono restituiti in modo asincrono, cioè passano attraverso la reply port del task. Nell'I/O sincrono, invece, lo stato di questo flag interviene più direttamente. In particolare determina il comportamento sincrono del comando BeginIO con i comandi immediati, e il possibile comportamento sincrono con i comandi normali (se infatti il QuickIO non viene accolto, la funzione diventa asincrona). Se invece si usa la funzione DoIO, impostare o non impostare IOF_QUICK è completamente indifferente dal momento che DoIO lo fa comunque.

Prima di passare ad altri argomenti, è bene ricordare a questo punto che sebbene si dica che "una richiesta di I/O viene inviata" oppure che "una struttura di I/O viene restituita alla tal coda", le strutture di I/O non vengono mai spostate dalle aree di RAM nelle quali sono state originariamente allocate. Sono i loro indirizzi che vengono copiati e manipolati all'interno del sistema.

Interazioni multiple fra task, reply port e dispositivi

La Figura 2.4 (nella pagina successiva) mostra tre task, ognuno contenente alcune chiamate alle funzioni di supporto alla libreria Exec. La figura non descrive l'ordine esatto d'esecuzione delle funzioni; mostra, invece, il modo in cui generalmente le funzioni di supporto alla libreria Exec lavorano con i dispositivi dell'Amiga.

Task2 è stato creato dalla funzione CreateTask contenuta in Task1, così come Task3 è stato creato, sempre con la funzione CreateTask, da Task2. Questo vuol dire che Task1 ha creato indirettamente Task3. In un esempio applicativo su una configurazione di questo tipo, Dispositivo1 potrebbe essere il dispositivo Serial collegato a un modem, Dispositivo2 il dispositivo Audio collegato a un altoparlante, e Dispositivo3 il dispositivo Printer collegato a una stampante attraverso la porta parallela dell'Amiga.

Per gestire in modo efficiente questi tre dispositivi, si presenta la necessità di creare tre task di gestione, uno per ogni dispositivo.

A seconda della particolare struttura del programma, la situazione può richiedere che Task1, Task2 e Task3 distinguano in due categorie i dati provenienti dai rispettivi dispositivi. In situazioni di questo tipo, per una più semplice gestione dei dati conviene che ciascun task si crei due strutture di I/O e due reply port.

Task1 contiene due chiamate alla funzione CreatePort, utilizzate per creare due reply port per ricevere le richieste di I/O restituite da Dispositivo1. Inoltre, Task1 contiene un certo numero di chiamate alla funzione CreateStdIO con lo scopo di creare strutture di tipo IOStdReq da utilizzare per impartire comandi a Dispositivo1; è presente una chiamata alla funzione CreateStdIO per creare la struttura IOStdReq da impiegare durante la chiamata alla funzione OpenDevice, una chiamata a CreateStdIO per ciascuna unità che si desidera aprire, e un corrispondente numero di chiamate alla funzione DeleteStdIO, che vengono utilizzate per eliminare le strutture IOStdReq quando non sono più necessarie. Infine, Task1 contiene una chiamata alla funzione CreateTask per

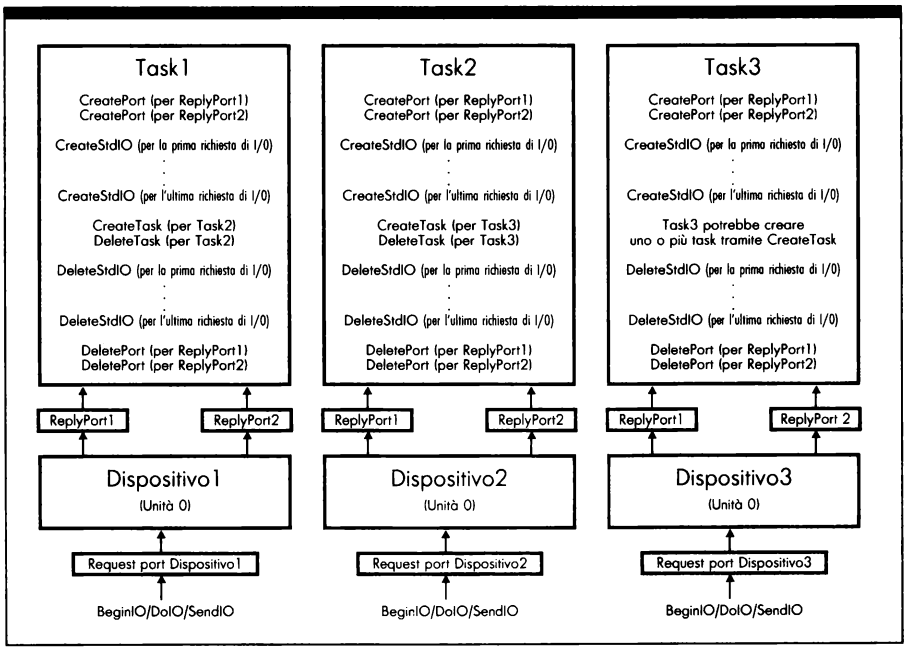


Figura 2.4: *Impiego delle funzioni di supporto alla libreria Exec per gestire task e reply port*

creare Task2, e una chiamata alla funzione DeleteTask per eliminare Task2 quando non è più necessario. Lo stesso schema di chiamate caratterizza anche Task2 e Task3.

Elaborazione delle richieste di I/O in modo immediato

Questa sezione analizza l'elaborazione delle richieste in modo immediato, un tipo di I/O al quale abbiamo già accennato illustrando l'I/O asincrono e sincrono. Acquistare familiarità con i comandi immediati è particolarmente semplice dal momento che operano tutti nello stesso modo.

In genere CMD_CLEAR, CMD_FLUSH, CMD_START, CMD_STOP e CMD_RESET vengono trattati automaticamente dai dispositivi come comandi immediati, ma non *sempre*: dipende soprattutto dalle caratteristiche del dispositivo.

Riassumendo tutte le considerazioni fatte finora sui vari modi in cui un comando può farsi strada attraverso il sistema dei dispositivi dell'Amiga, si giunge alla conclusione che una richiesta di I/O può seguire soltanto quattro possibili percorsi.

1. Un comando può essere inviato all'unità di un dispositivo, e quindi essere accodato nella relativa coda alla request port; il dispositivo elabora il comando quando giunge il suo turno, e lo restituisce al task inserendolo nella coda alla reply port del task. Questo percorso comprende accodamenti alla fine di entrambi i passaggi (l'abbiamo incontrato descrivendo l'I/O asincrono dei comandi non immediati, e l'I/O pseudo-sincrono che si verifica quando il QuickIO non viene accolto, ma il task ha inviato la richiesta tramite la funzione DoIO).

2. Un comando può essere inviato all'unità di un dispositivo impostando nel parametro io_Flags della struttura di I/O il flag IOF_QUICK. Se il QuickIO viene accordato (perché le condizioni in cui si trova il sistema lo consentono) le routine interne del dispositivo elaborano il comando immediatamente e non lo restituiscono al task. Il flag IOF_QUICK risulta ancora impostato nella risposta alla richiesta di I/O. In questo tragitto la richiesta non viene mai accodata (l'abbiamo incontrato descrivendo l'I/O sincrono dei comandi non immediati, che il task può effettuare tramite la funzione BeginIO con il flag IOF_QUICK impostato e DoIO).

3. Un comando può essere inviato impostando il flag IOF_QUICK, ma il QuickIO non viene accordato (perché il comando, per esempio, non lo consente). In questo caso la richiesta di I/O viene automaticamente accodata nella coda alla request port dell'unità, e le routine interne del dispositivo la elaborano quando viene il suo turno; a elaborazione ultimata, la richiesta viene inserita nella coda alla reply port del task che ha inviato il comando. Anche questo percorso (come il primo) prevede l'accodamento dei dati a entrambe le code. Il flag IOF_QUICK nel parametro io_Flags della struttura di I/O restituita come risposta

risulterà azzerato. Questo tragitto viene considerato asincrono se il comando è stato inviato tramite la funzione BeginIO, e sincrono se il comando è stato inviato tramite la funzione DoIO.

4. Alcuni particolari comandi, infine, vengono automaticamente eseguiti dai dispositivi in modo immediato: i dispositivi sono programmati per eseguirli sempre immediatamente, con precedenza assoluta e indipendentemente dalla situazione in cui si trova il sistema. Questi comandi possiedono un'alta priorità d'esecuzione, e vengono restituiti alla reply port dei task che li hanno inviati solo se questi non hanno impostato il flag IOF_QUICK. Se il comando è stato inviato tramite la funzione DoIO, agli occhi del task l'esecuzione immediata è del tutto equivalente all'esecuzione sincrona. Se invece è stata usata la funzione BeginIO, l'esecuzione sarà sincrona solo se il task ha impostato il flag IOF_QUICK. Comunque si noti che nel momento in cui il task riprende il controllo può essere sempre certo (anche se il flag non è stato impostato) che il comando è già stato eseguito. L'unica vera differenza con il comportamento sincrono riguarda il fatto che nel caso immediato ma non sincrono il task deve preoccuparsi di estrarre la risposta dalla coda alla sua reply port.

Illustriamo l'ultimo caso con un esempio. Si consideri un task che ha inviato il comando immediato CMD_RESET a un dispositivo al fine di ripristinare in una delle sue unità lo stato di default. Il comando non viene accodato e il dispositivo lo esegue immediatamente, anche se l'unità stava elaborando un altro comando. Poi lo restituisce alla reply port del mittente se il flag IOF_QUICK è azzerato (altrimenti non lo restituisce). Con quanto già detto risulta ovvio che se è stata impiegata la funzione DoIO, la risposta non viene mai accodata alla reply port del task, mentre la possibilità di accodamento esiste se è stata impiegata la funzione BeginIO.

Questa elevata priorità dei comandi immediati può creare talvolta dei problemi. Se per esempio viene inviato il comando immediato CMD_STOP a un'unità che sta eseguendo un comando CMD_WRITE, l'esecuzione del comando CMD_WRITE viene subito interrotta, e questo significa che alcuni byte non vengono trasmessi.

I dettagli operativi sui comandi immediati variano da dispositivo a dispositivo. Le analisi dei comandi presenti nei capitoli che seguono indicheranno esplicitamente quando un comando è di tipo immediato, anche se per il programmatore non è di reale interesse, dal momento che impiegando la funzione DoIO il comportamento del comando è in pratica sempre uguale, con l'eccezione di qualche breve periodo d'attesa. L'unico caso in cui è necessaria BeginIO è quando il dispositivo prevede comandi immediati che prendono in considerazione altri flag del parametro io_Flags.

Procedure generali di gestione delle richieste di I/O

Questa sezione analizza i procedimenti da seguire per inizializzare e gestire le strutture per le richieste di I/O. Queste fasi possono essere applicate a ciascun dispositivo dell'Amiga.

Dal momento che la struttura IORequest è la struttura minima necessaria per l'invio di richieste di I/O alle unità di un dispositivo, le specifiche strutture di I/O di ogni dispositivo la includono sempre come primo elemento. Questa struttura contiene una sotto-struttura Message contenente il parametro mn_ReplyPort, i puntatori io_Device e io_Unit, i parametri io_Command, io_Flags e io_Error. Supponendo che la struttura IORequest sia già stata allocata dal task tramite le istruzioni in C:

```
struct IORequest *iORequest;
...
    iORequest = (struct IORequest *)AllocMem
        ((long)sizeof(struct IORequest), MEMF_PUBLIC | MEMF_CLEAR);
...
```

vediamo quali sono le operazioni da compiere per inizializzare i suoi parametri.

- mn_ReplyPort. Questo parametro dev'essere inizializzato in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si tratta della message port che riceverà la richiesta di I/O quando questa verrà restituita. Le istruzioni in C che si possono inserire nei propri sorgenti per inizializzare il parametro mn_ReplyPort sono le seguenti:

```
struct MsgPort *CreatePort();
...
    iORequest->io_Message.mn_ReplyPort =
        CreatePort ("Nome Porta", 0L);
...
```

con le quali creiamo message port pubbliche e con priorità pari a zero. Si noti che il meccanismo per la gestione delle risposte viene controllato automaticamente dalle routine interne del dispositivo. Il parametro mn_ReplyPort può puntare a *qualsiasi* struttura MsgPort appartenente a *qualsiasi* task nel sistema, oppure può addirittura contenere il valore 0, nel qual caso il messaggio non viene restituito. Specificando mn_ReplyPort in modo che punti alla struttura MsgPort di un altro task, il task che invia la richiesta riesce a fare in modo che i dati restituiti dal dispositivo giungano a quel task. Questo è un altro sistema per trasferire i dati generati dalle routine interne del dispositivo tra i diversi task. Immaginando le routine interne del dispositivo come un terzo task, si ottiene un intreccio di legami composto da un task che invia i dati a un altro task utilizzando un terzo task (il dispositivo) per generarli. Le istruzioni in C che inizializzano mn_ReplyPort con l'indirizzo di una

message port pubblica sono le seguenti:

```
...
iORequest->io.Message.mn_ReplyPort = FindPort("Nome Porta");
```

dove la stringa "Nome Porta" individua il nome della message port associata al task che dovrà ricevere le risposte inoltrate dal dispositivo.

Se un task utilizza le funzioni CreateStdIO oppure CreateExtIO, il parametro mn_ReplyPort viene aggiornato automaticamente con l'indirizzo della message port indicata dal task come argomento. Inoltre, le funzioni CreateStdIO e CreateExtIO inizializzano i parametri ln_Type e ln_Pri contenuti nella sotto-struttura Node della struttura Message impiegata nella richiesta di I/O. Quindi, se un task impiega queste due funzioni per creare le strutture per le proprie richieste di I/O, non deve preoccuparsi d'inizializzare questi due parametri. Vediamo un esempio con la funzione CreateStdIO.

```
struct MsgPort *msgPort;
struct MsgPort *CreatePort();
struct IOStdReq *iOStdReq;
struct IOStdReq *CreateStdIO();
...
     msgPort = CreatePort("Nome Porta", ØL);
     iOStdReq = CreateStdIO(msgPort);
...
```

- io_Device. È un puntatore a una struttura Device e la sua funzione è indicare a quale dispositivo si desidera inviare la richiesta. Ogni volta che il task chiama la funzione OpenDevice per aprire un'unità, il parametro io_Device viene impostato con l'indirizzo della struttura Device che definisce quel dispositivo. Se il task desidera allocare e impiegare altre strutture di I/O per inviare comandi all'unità aperta, è necessario che vi copi i parametri io_Device e io_Unit ottenuti aprendo l'unità del dispositivo con la funzione OpenDevice. Si ricordi infine che la struttura Device è unica per il dispositivo e per tutti i task che accedono alle sue unità, e rimane allocata finché il dispositivo rimane disponibile.

- io_Unit. È un puntatore a una struttura Unit e la sua funzione è indicare a quale unità si desidera inviare la richiesta. Ogni volta che il task chiama la funzione OpenDevice per aprire un'unità di un dispositivo, questa funzione memorizza nel parametro io_Unit l'indirizzo della struttura Unit che definisce quell'unità. Ogni unità possiede una propria struttura Unit: se più task aprono la stessa unità, nelle loro strutture di I/O i parametri io_Unit vengono a contenere l'indirizzo della stessa struttura Unit. Questa struttura contiene una message port, la nota request port, nella cui coda vengono inserite le richieste di I/O dirette all'unità. Ogni volta che il task si appresta a inviare una richiesta di I/O

a una particolare unità che ha già aperto, deve assicurarsi che nel parametro io_Unit sia presente l'indirizzo della corrispondente struttura Unit. La necessità di questo controllo è evidente soprattutto quando un task apre più unità di uno stesso dispositivo, ma utilizza la stessa richiesta di I/O per tutte.

- io_Command. Dev'essere inizializzato dal task con il codice numerico corrispondente al comando che desidera inviare all'unità del dispositivo. Il comando dev'essere ovviamente uno di quelli riconosciuti dal dispositivo, pena la restituzione di un codice d'errore. I file INCLUDE assegnano un valore specifico a ogni comando che il dispositivo può eseguire.

- io_Flags. È composto da un insieme di flag che descrivono a un'unità le particolari necessità del task che ha inviato la richiesta di I/O. In alcuni casi un task può inizializzare questo parametro prima di aprire il dispositivo con la funzione OpenDevice. Per esempio, se un task vuole aprire il dispositivo nel modo di accesso condiviso, deve specificarlo impostando l'opportuno bit del parametro io_Flags nella prima struttura di I/O che utilizzerà per effettuare la chiamata a OpenDevice. Alcuni dispositivi vengono aperti automaticamente nel modo di accesso esclusivo, a meno che il parametro io_Flags non dia indicazioni diverse. Per ogni singolo dispositivo potrebbero esservi altri flag da specificare durante le chiamate alla funzione OpenDevice; nei capitoli seguenti vengono analizzati tutti i flag riconosciuti da ciascun dispositivo.

  Una volta definita la struttura di I/O necessaria per aprire il dispositivo, è possibile specificare altre configurazioni del parametro io_Flags utili per le successive richieste. Per esempio, un task può impostare il flag IOF_QUICK del parametro io_Flags per i comandi che consentono l'I/O veloce. Si ricordi però che quando s'imposta qualche flag, è necessario inviare la richiesta di I/O tramite la funzione BeginIO, la quale non altera il contenuto del parametro io_Flags.

- io_Error. Il valore di questo parametro viene normalmente stabilito dalle routine interne del dispositivo prima di restituire la richiesta di I/O al task che l'aveva inoltrata. Indica l'esito ottenuto dalla richiesta. I vari codici d'errore che ogni dispositivo può restituire verranno discussi nei prossimi capitoli.

Classi delle richieste di I/O

Tutte le richieste di I/O rientrano nelle due classi seguenti.

1. *Quelle definite da una struttura IOStdReq*. Questa struttura è costituita da una sotto-struttura IORequest come primo elemento e da quattro ulteriori parametri (io_Actual, io_Length, io_Data e io_Offset). Il parametro io_Data permette a un task d'indicare l'indirizzo di un'area

di memoria che può essere utilizzata come destinazione e/o sorgente delle informazioni che provengono dalle routine interne del dispositivo o che verso di esse sono dirette (la struttura IORequest, al contrario, non permette a un task e a un dispositivo di comunicare fra loro tramite i buffer del task in quanto nessuno dei suoi parametri viene impiegato come puntatore a un buffer; ecco la necessità d'impiegare la struttura IOStdReq, estensione standard della struttura IORequest).

2. *Quelle definite da strutture di I/O non standard*, cioè specifiche per ogni dispositivo. Un esempio è costituito dal dispositivo TrackDisk, che richiede ai task d'impiegare la struttura IOExtTD per comunicare con le sue unità. Le strutture di I/O non standard previste da ciascun dispositivo sono riassunte nella Tavola 1.2 del capitolo 1 (a pagina 32).

Creazione di richieste multiple

Se si desidera comunicare con un dispositivo, occorre accertarsi che la struttura di I/O che ci si appresta a impiegare non sia già in uso per una richiesta non ancora restituita (infatti, se così fosse, si rischierebbe di alterarne il contenuto compromettendo la precedente richiesta: una delle prime regole del sistema multitasking dell'Amiga è che quando una richiesta, o più in generale un messaggio, è stato inviato a un task o a un dispositivo, questo se ne appropria e assume che nessun altro task lo altererà fino a quando non sarà lo stesso task o dispositivo a restituirlo). Questo paragrafo presenta le regole da osservare quando si procede alla creazione di due o più strutture di I/O per definire i dati necessari ai propri task.

Un task può ottenere le strutture di I/O di cui ha bisogno in due modi: può crearne di nuove con le funzioni CreateStdIO o CreateExtIO e inizializzarle chiamando la funzione OpenDevice se deve anche aprire un'unità, o copiando i parametri delle strutture di I/O ottenute con una precedente chiamata a OpenDevice. Oppure può riutilizzare, una volta che hanno terminato il ciclo dal task al dispositivo e viceversa, le strutture già definite cambiandone semplicemente i parametri. Il procedimento più consigliabile dipende dal punto specifico all'interno del task nel quale nasce l'esigenza di disporre di una struttura per le richieste di I/O, da quali strutture di I/O sono già definite in quel punto, e da quello che si vuole ottenere.

Se una struttura di I/O adibita a comunicare con una particolare unità viene inizializzata ex novo tramite OpenDevice per aprire un'altra unità, questa funzione inizializza prima di tutto i parametri io_Device e io_Unit in modo che puntino rispettivamente alla struttura Device del dispositivo e alla nuova struttura Unit. Una volta compiuta l'inizializzazione di questi due parametri, un task può copiarli in altre strutture di I/O correttamente allocate.

Inoltre, la stessa struttura di I/O che viene inizializzata da OpenDevice può essere impiegata più volte con le funzioni BeginIO, DoIO e SendIO; l'unica condizione richiesta è che la struttura in questione non sia già in uso. Quando il task riottiene una particolare struttura di I/O che aveva inviato, i parametri in essa contenuti (io_Flags, io_Data. io_Length e così via) possono essere

ridefiniti per inviare nuove richieste di I/O alla stessa unità (io_Device e io_Unit inalterati), a un'altra unità del dispositivo (io_Device inalterato, io_Unit alterato), all'unità di un altro dispositivo (io_Device e io_Unit entrambi alterati).

Elaborazione di più richieste di I/O

La Figura 2.5 (nella pagina successiva) mostra una task di gestione di un dispositivo in esecuzione. Questa figura rappresenta l'interazione tra task e dispositivo, ed è valida per ognuno dei 12 dispositivi dell'Amiga. Per esempio, il grosso rettangolo potrebbe rappresentare le istruzioni di un task che si occupa della gestione dei dati su disco tramite un'unità del dispositivo TrackDisk.

I piccoli rettangoli all'interno del rettangolo maggiore rappresentano strutture di I/O con le relative operazioni compiute dal task per definirle. Ognuna di queste strutture può essere creata con le funzioni CreateStdIO o CreateExtIO, le quali allocano la memoria necessaria, inizializzano i parametri ln_Type e ln_Pri della struttura Node contenuta nella struttura io_Message, e il parametro mn_ReplyPort con il mittente indicato dal task. Gli altri parametri, come io_Data, io_Length, io_Actual, io_Offset e così via, sono invece di competenza del task.

Le strutture di I/O potrebbero anche essere create con un metodo di più basso livello, ovvero utilizzando direttamente le funzioni della libreria Exec e le istruzioni di assegnazione. Questa via è però sconsigliata, in quanto le funzioni di supporto della libreria Exec sono facili da usare e riducono le dimensioni dei codici sorgente.

Nell'esempio illustrato dalla figura, una volta che la richiesta IORequest0 è stata parzialmente inizializzata dal task con uno dei due metodi appena descritti, il suo indirizzo viene passato come argomento alla funzione OpenDevice per aprire l'unità del dispositivo; OpenDevice si preoccupa di rendere disponibile il dispositivo qualora non lo sia, e d'impostare opportunamente i parametri io_Device e io_Unit.

A questo punto, la struttura IORequest0 è stata completamente inizializzata, ed è pronta per essere usata nelle richieste di I/O. Al task non rimane che aggiornare i parametri che descrivono la richiesta (per esempio il parametro io_Command) e inoltrarla tramite una delle due funzioni asincrone BeginIO o SendIO. L'invio di questa richiesta viene mostrato in figura tramite la freccia che dal rettangolo più grande (che rappresenta il task di gestione del dispositivo) scende verso il fondo della figura per giungere alla coda della request port.

Se l'unità del dispositivo non ha nessuna richiesta di I/O accodata, la richiesta inviata dal task va a occupare direttamente la sommità della coda. Se invece l'unità possiede già altre richieste nella propria coda, la richiesta IORequest0 verrà collocata all'ultimo posto. Le routine interne del dispositivo procedono alla sua elaborazione solo quando raggiunge la sommità.

Dopo aver inoltrato la richiesta, il task riottiene subito il controllo avendo usato l'I/O asincrono, e può proseguire nello svolgimento di altre mansioni. Supponiamo che debba inviare altre richieste di I/O alla stessa unità dello

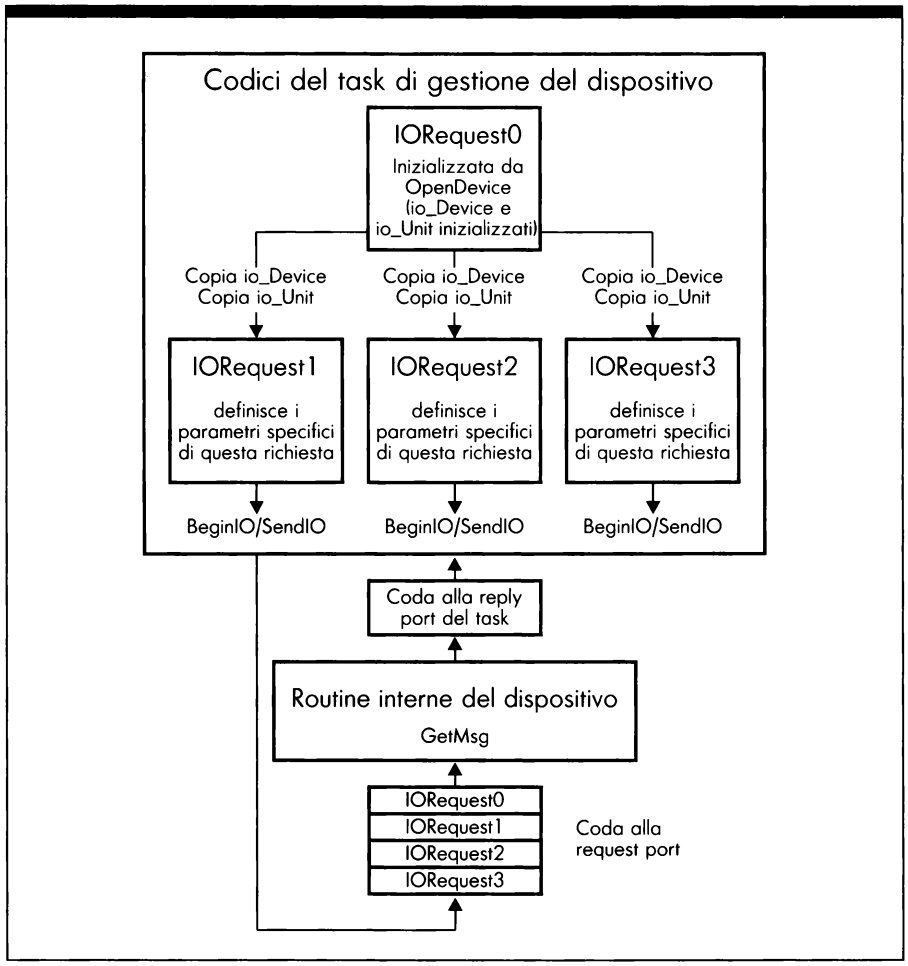


Figura 2.5: *Invio di diverse richieste di I/O a un dispositivo*

stesso dispositivo. Per farlo, com'è già stato spiegato, non può utilizzare ancora la struttura IORequest0, dal momento che questa è già in uso. Il task deve quindi creare una serie di altre richieste di I/O per soddisfare le proprie necessità di dati, allocando e inizializzando opportunamente strutture di I/O addizionali. Nella figura, IORequest1, IORequest2 e IORequest3 rappresentano le nuove strutture di I/O, che il task crea ancora una volta chiamando le funzioni CreateStdIO o CreateExtIO. Supponendo che desideri usare una sola reply port, il task deve indicare in queste strutture la message port già allocata per la prima richiesta, e deve copiare nei parametri io_Device e io_Unit gli indirizzi ottenuti quando ha chiamato OpenDevice sempre con la prima richiesta.

Dato che si tratta di richieste di I/O asincrono, il task le può inviare una di seguito all'altra senza attendere le risposte. Quando lo ritiene opportuno, può impiegare CheckIO o WaitIO per iniziare a esaminare i risultati restituiti dall'unità.

La situazione descritta potrebbe essere quella di un task che ha necessità di compiere accessi in lettura a quattro file diversi su uno stesso disco, ma può compiere altre operazioni anche se i dati non sono ancora pervenuti e preferisce quindi non entrare in attesa (I/O sincrono).

Se invece il task deve accedere in lettura ai dati di un file di cui non può fare a meno per continuare il suo lavoro, allora utilizza la funzione sincrona DoIO per comunicare con l'unità. La differenza fondamentale rispetto al caso asincrono è che con DoIO il task può inviare un nuovo comando solo quando ha ricevuto risposta al precedente, e quindi può usare per ogni comando sempre la stessa struttura di I/O. Inoltre, se il task è strutturato in modo da non poter continuare se non riceve ogni volta i dati richiesti, la soluzione sincrona svolge una funzione benefica nei confronti dell'intero sistema, permettendo ad altri task di ricevere il controllo con maggiore frequenza, ed evitando i loop di attesa, così dannosi in un sistema multitasking.

Si noti infine che questa discussione è stata sviluppata descrivendo il caso di un solo task e una sola unità di un dispositivo, ma può essere facilmente estesa a più task, più reply port, diverse unità e diversi dispositivi.

Le funzioni di accesso ai dispositivi

Vengono impiegate sei funzioni per controllare la maggior parte delle operazioni di I/O: WaitIO, CheckIO, DoIO, SendIO e AbortIO, della libreria Exec, e BeginIO della libreria di ogni dispositivo. Per chiamare quest'ultima, nonostante costituisca una routine interna del dispositivo, si opera come per chiamare una routine dell'Exec. Inoltre, nella libreria di ogni dispositivo è presente anche la funzione AbortIO, che può essere chiamata tramite l'omonima funzione della libreria Exec, la quale accede alla libreria del dispositivo per chiamarla a sua volta.

Dato che AbortIO e BeginIO ricorrono nelle librerie di ciascun dispositivo, e la loro struttura interna è simile per ogni dispositivo, questa sezione ne illustra le caratteristiche comuni e gli impieghi. Ogni specifica differenza introdotta da un dispositivo rispetto a questa trattazione generale verrà evidenziata nei capitoli che seguono. Seguiranno poi le descrizioni dettagliate delle altre quattro funzioni dell'Exec.

La funzione AbortIO

AbortIO sopprime una particolare richiesta di I/O inviata a un'unità di uno dei 12 dispositivi dell'Amiga (in questo contesto, con le parole "soppressione della richiesta" intendiamo semplicemente l'annullamento della sua operatività, e non la sua disallocazione dalla memoria). AbortIO è in grado di sopprimere sia le richieste attive (cioè in fase di elaborazione da parte dell'unità), sia quelle ancora accodate (cioè in attesa di essere elaborate). Se la richiesta risulta accodata, la funzione la rimuove dalla coda alla request port dell'unità, ne imposta il parametro io_Error con il codice d'errore IOERR_ABORTED e la

restituisce al mittente se per essa non è stato richiesto il QuickIO. Se invece la richiesta di I/O è già in corso di elaborazione, la funzione AbortIO blocca l'esecuzione del comando alla prima occasione, ma non memorizza alcun codice d'errore nel parametro io_Error.

Dopo l'esecuzione del comando AbortIO, il task può analizzare la struttura di I/O inviatagli dall'unità e leggere il valore assunto da io_Error, comportandosi di conseguenza. In particolare, il task può modificare la struttura di I/O e inviarla nuovamente al dispositivo perché la elabori.

Al contrario del comando CMD_FLUSH, la funzione AbortIO fornisce un meccanismo per abortire una singola richiesta di I/O inviata da un task a un'unità di un dispositivo.

La funzione BeginIO

BeginIO è la funzione più importante di ogni dispositivo, dal momento che è sempre lei a ricevere le richieste di I/O inviate al dispositivo, anche quando il task impiega le funzioni DoIO o SendIO dell'Exec (come vedremo più avanti). Nella libreria Exec non esiste una funzione BeginIO, e quindi quando i task la chiamano cedono direttamente il controllo all'omonima funzione del dispositivo che devono aver già aperto.

BeginIO ha comportamenti diversi a seconda del dispositivo. Per esempio, se il dispositivo dispone di diverse unità, la funzione accerta che il comando non sia immediato e quindi accoda la richiesta di I/O alla request port relativa all'unità indirizzata. Se invece il dispositivo dispone di una sola unità, questa selezione non viene effettuata, dal momento che la coda alla request port è unica.

Quando i task accedono direttamente alla funzione BeginIO anziché passare da DoIO o SendIO, possono impostare anche tutti i flag del parametro io_Flags previsti dal dispositivo: BeginIO ne prende visione e si comporta di conseguenza. È quindi particolarmente idonea per comunicare con i dispositivi Audio e Serial, che riconoscono anche altri flag, oltre a IOF_QUICK. DoIO e SendIO non sarebbero altrettanto adatte dal momento che entrambe azzerano tutti i flag (e DoIO imposta sempre IOF_QUICK).

Analizziamo il funzionamento di tutte le funzioni BeginIO presenti nei vari dispositivi dell'Amiga, iniziando dal punto di vista di chi le chiama (potrebbe trattarsi di un task o del sistema). Qualsiasi sia il tipo di comando (immediato o normale), se il task imposta il flag IOF_QUICK nella richiesta, non appena riottiene il controllo può controllare lo stato del flag: se risulta azzerato significa che il comando verrà restituito alla reply port del task quando la sua elaborazione sarà terminata. Il task riottiene il controllo quando il comando non è ancora stato elaborato, e quindi non deve accedere alla relativa struttura di I/O (se non per leggere lo stato del flag IOF_QUICK) fino a quando il dispositivo non la restituisce.

Se invece il flag IOF_QUICK risulta ancora impostato, il task deve desumere che il comando è stato eseguito in maniera sincrona, e che alla sua reply port non è stata accodata nessuna risposta; può quindi servirsi della stessa struttura di I/O per inoltrare altre richieste. Questa descrizione considera

la funzione BeginIO come una misteriosa scatola nera di cui conosciamo soltanto gli effetti, e si applica a qualsiasi tipo di comando. Vediamo ora cosa accade all'interno della funzione, analisi che ci permetterà di comprendere a fondo il significato di accodamento, QuickIO ed esecuzione immediata.

Quando BeginIO riceve il controllo, si aspetta di trovare l'indirizzo della richiesta di I/O nel registro A1 della CPU. La prima operazione che compie è andare ad azzerare il parametro io_Error. Subito dopo imposta il parametro ln_Type della richiesta a NT_MESSAGE e verifica se il comando indicato nel parametro io_Command rientra fra quelli riconosciuti dal dispositivo. Se non rientra, imposta il parametro io_Error a IOERR_NOCMD e restituisce il controllo. Se invece il comando è eseguibile, il flusso di operazioni segue due strade a seconda che il comando sia immediato o normale.

Se il comando è immediato, BeginIO lo esegue istantaneamente agendo sull'unità interessata, e prima di restituire il controllo chiama la funzione ReplyMsg (ma solo se il flag IOF_QUICK risulta azzerato).

Se invece il comando è normale, BeginIO verifica se è stato richiesto il QuickIO. Se la verifica è positiva, la funzione rileva lo stato del dispositivo e decide se la modalità di accesso veloce è accordabile. In caso affermativo, la richiesta non viene accodata alla request port dell'unità indirizzata, e il comando viene eseguito istantaneamente. A esecuzione ultimata, BeginIO restituisce semplicemente il controllo senza accodare la risposta alla reply port del task; il flag IOF_QUICK non viene azzerato. Se invece il QuickIO non viene accordato, oppure se non era stato richiesto, la funzione accoda la richiesta alla request port dell'unità interessata, ne azzera il flag IOF_QUICK e restituisce il controllo al task. In questo caso, l'esecuzione del comando è di tipo asincrono, in quanto BeginIO restituisce il controllo quando l'elaborazione del comando non solo non si è ancora conclusa, ma non è neanche iniziata; il task riceverà la risposta alla sua reply port solo quando il comando sarà stato eseguito.

Questa è la struttura generale della funzione BeginIO. Ovviamente, lo svuotamento della coda alla request port e l'esecuzione dei comandi non immediati sono operazioni che vengono condotte da un task del dispositivo del tutto indipendente da BeginIO. La separazione dei ruoli è evidente: BeginIO filtra tutti i comandi immediati, mentre con i comandi normali per i quali il QuickIO è possibile attende che vengano eseguiti prima di restituire il controllo al task.

Il task interno del dispositivo, invece, accede periodicamente alle message port delle varie unità del dispositivo per prelevare le eventuali richieste accodate e iniziarne l'elaborazione. Ogni volta che un comando è stato interamente eseguito, il task interno ne imposta il parametro ln_Type a NT_REPLYMSG e chiama la funzione ReplyMsg per accodare la richiesta alla reply port indicata nel parametro mn_ReplyPort. Soltanto quando è stata eseguita anche questa operazione il task mittente può tornare a impiegare la struttura di I/O e leggerne i parametri. Nell'analisi del comando DoIO vedremo che il parametro ln_Type serve anche per distinguere tra loro le varie richieste di I/O giunte alla reply port del task.

La funzione sincrona DoIO

Questa funzione è contenuta nella libreria Exec e viene utilizzata dai task quando desiderano inviare le richieste ai dispositivi in modo sincrono. Prima di analizzare la sua struttura interna, vediamo come appare agli occhi dei task quando la impiegano. Di qualunque tipo sia il comando, e che venga accordato o meno il QuickIO, con DoIO il task ha la certezza che quando riottiene il controllo il comando è stato eseguito (o è stato restituito con una condizione d'errore) e non si trova accodato alla sua reply port. Si tratta di un funzionamento che evita ai task numerosi controlli. Vediamo come viene ottenuto.

Quando DoIO riceve il controllo, si aspetta che nel registro A1 sia presente l'indirizzo della richiesta di I/O da inoltrare al dispositivo. La prima operazione che compie è impostare il flag IOF_QUICK nel parametro io_Flags della struttura di I/O, e azzerare tutti gli altri. Questa operazione fa sì che la struttura di I/O venga inoltrata richiedendo il QuickIO, ma azzera ogni altro flag che il task potrebbe aver impostato e che il dispositivo potrebbe controllare. Per questa ragione, DoIO non è idonea a inviare richieste ai dispositivi Audio e Serial, e in generale a tutti quei dispositivi che accedono anche agli altri flag del parametro io_Flags, oltre che a IOF_QUICK.

Successivamente, DoIO estrae dalla struttura di I/O l'indirizzo contenuto nel parametro io_Device, l'indirizzo base della struttura Library di gestione della libreria del dispositivo, e cede il controllo alla funzione di offset −30, BeginIO. In questo modo, DoIO cede il controllo proprio alla funzione BeginIO contenuta nel dispositivo indicato dalla richiesta.

Quando DoIO riottiene il controllo, verifica lo stato del flag IOF_QUICK che aveva precedentemente impostato: se è ancora impostato, significa che il comando è stato eseguito, e che la richiesta di I/O non è stata accodata alla reply port del task. In questo caso, DoIO memorizza nel registro D0 il valore contenuto nel parametro io_Error e restituisce il controllo. Se invece il flag risulta azzerato, significa che il dispositivo tratta la richiesta in maniera asincrona. DoIO allora entra in un ciclo che chiama la funzione Wait dell'Exec indicando il bit di segnale allocato alla reply port del task, e ogni volta che giunge un segnale relativo a quella message port verifica se nel parametro ln_Type della struttura di I/O è presente la costante NT_REPLYMSG. Se l'esito è positivo significa che il messaggio giunto alla reply port del task è proprio la richiesta di I/O di cui DoIO si sta prendendo cura, mentre se l'esito è negativo DoIO rientra in attesa chiamando di nuovo la funzione Wait.

Quando DoIO rileva che la richiesta è stata finalmente accodata alla reply port del task, la rimuove dalla coda e restituisce il controllo al task. Volendo, il task può rendersi conto che la richiesta è stata trattata in maniera asincrona e accodata alla sua reply port rilevando che il flag IOF_QUICK è azzerato.

Come si può notare, DoIO è una funzione abbastanza complessa, in grado di rendere pseudo-sincrono l'I/O anche quando il dispositivo tratta la richiesta in modo asincrono.

La funzione asincrona SendIO

Questa funzione è contenuta nella libreria Exec e viene utilizzata dai task quando desiderano inviare le richieste ai dispositivi in modo asincrono. Il task che la chiama ha la certezza di riottenere subito il controllo e sa che difficilmente il comando è già stato eseguito, a meno che non si tratti di un comando immediato. In ogni caso, il task può procedere allo svolgimento di altri compiti prima di controllare se alla sua reply port è giunta risposta, cioè se il comando è stato elaborato.

Internamente, la funzione SendIO è davvero semplicissima: compie infatti due sole operazioni. Azzera completamente il parametro io_Flags della richiesta di I/O, estrae l'indirizzo base della libreria dal parametro io_Device, e chiama la funzione BeginIO. Dal momento che si tratta di una funzione asincrona, quando riottiene il controllo non compie altre operazioni e cede il controllo al task. Quindi, assomiglia a DoIO in quanto azzera tutti i flag del parametro io_Flags, e se ne distacca perché non imposta il flag IOF_QUICK. Ricordando quanto abbiamo detto a proposito della funzione BeginIO, questo significa che, a prescindere dal tipo di comando (normale o immediato), la risposta alla richiesta di I/O verrà sempre accodata alla reply port del task.

I task dovrebbero impiegare SendIO tutte le volte che intendono inviare un comando in modo asincrono e il dispositivo non prende in considerazione altri flag oltre IOF_QUICK. Se invece si presenta la necessità d'indicare uno o più di questi flag, occorre chiamare la funzione BeginIO avendo l'accortezza di azzerare il flag IOF_QUICK (operando in questo modo si replica quasi esattamente il comportamento di SendIO).

La funzione sincrona WaitIO

Questa funzione permette a un task di entrare in attesa che una particolare richiesta di I/O, quella indicata come argomento della funzione, giunga alla sua reply port. Prima di analizzarne il funzionamento, è bene sottolineare che al pari della funzione DoIO presenta qualche rischio, in quanto se per qualche ragione la richiesta indicata non viene più restituita dal dispositivo, il task entra in un'attesa perenne. Dopo averne illustrato il funzionamento, descriveremo un metodo che potrebbe sbloccare la situazione in un simile caso.

Internamente, la funzione WaitIO è del tutto simile alla seconda parte della funzione DoIO, quella che segue la chiamata della funzione BeginIO. Quando ottiene il controllo si aspetta che nel registro A1 della CPU sia memorizzato l'indirizzo della richiesta da attendere, quello indicato nella chiamata della funzione come argomento. La prima operazione che viene effettuata è verificare lo stato del flag IOF_QUICK nel parametro io_Flags della richiesta. Se il flag risulta impostato WaitIO restituisce subito il controllo copiando nel registro D0 il valore contenuto nel parametro io_Error. Questo valore sarà quello che verrà memorizzato nella variabile eguagliata alla funzione; in questo caso non si è verificato alcun accodamento della risposta.

Se invece il flag risulta azzerato, WaitIO entra in un ciclo all'interno del quale chiama la funzione Wait indicando il bit di segnale allocato alla reply port

del task; ogni volta che riottiene il controllo (è giunto un messaggio nella reply port) controlla se il parametro ln_Type della richiesta che sta aspettando contiene la costante NT_REPLYMSG. Se l'esito è negativo rientra nel loop chiamando nuovamente Wait, mentre se è positivo significa che il dispositivo ha restituito la richiesta di I/O attesa. In questo secondo caso, WaitIO provvede a estrarla dalla coda alla reply port del task e a restituire il controllo memorizzando nel registro D0 il contenuto del parametro io_Error. Come si può notare, WaitIO permette di trasformare in pseudo-sincrono un I/O che è stato avviato come asincrono tramite SendIO. Al pari di DoIO, WaitIO provvede sempre a rimuovere la risposta dalla coda alla reply port del task.

In fase di debug, se per qualche ragione il task non riottiene il controllo si può intervenire dall'esterno simulando la restituzione della richiesta, a patto però che il task abbia effettuato due operazioni che ovviamente interessano solo il debug: prima d'inviare la richiesta di I/O e di chiamare WaitIO, deve aver allocato un'altra message port pubblica, dotata di un particolare nome proprio (per esempio, "IORequest"), alla quale ha inviato un messaggio nel cui parametro io_Length ha memorizzato l'indirizzo della richiesta di I/O da inviare al dispositivo e della quale desidera entrare in attesa chiamando la funzione WaitIO. Compiute queste due operazioni, può procedere a chiamare WaitIO.

Se ora il programmatore rileva che il task non si riprende dall'attesa, può creare un secondo task che effettui le seguenti operazioni. Tramite il nome proprio citato e la funzione FindPort ottiene l'indirizzo della message port di debug allocata come pubblica dal primo task. Successivamente, vi accede per prelevare il messaggio che in essa si aspetta di trovare, e quindi ricava dal parametro io_Length l'indirizzo della richiesta di I/O che non è stata restituita. Disponendo di questo indirizzo, accede al parametro ln_Type della richiesta e vi memorizza la costante NT_REPLYMSG. Successivamente, chiama la funzione ReplyMsg per restituire questo messaggio al mittente, e come per magia il task addormentato riottiene il controllo.

Ovviamente, questo semplice metodo si può applicare anche quando il task entra in attesa con la funzione DoIO e l'attesa sembra diventare eterna.

Riguardo a WaitIO si noti che chiamando SendIO e successivamente WaitIO si ricalca in pratica il comportamento della funzione DoIO, con la differenza che il QuickIO non viene richiesto.

La funzione asincrona CheckIO

CheckIO consente ai task di scoprire se una particolare richiesta di I/O inoltrata a un dispositivo è stata restituita. A differenza di WaitIO, questa funzione non entra in attesa se la richiesta non risulta ancora restituita. I task la chiamano eguagliandola a una variabile che dovranno poi leggere per ottenere il risultato della verifica. Se la variabile contiene un valore nullo, significa che la richiesta di I/O indicata come argomento non è ancora pervenuta. Se invece la variabile contiene un valore diverso da 0, significa che la richiesta è stata restituita: in questo caso CheckIO restituisce nella variabile l'indirizzo della richiesta, che ovviamente coincide con quello indicato dal task come argomento della funzione. Tramite CheckIO i task possono periodicamen-

te verificare se la richiesta di I/O è stata soddisfatta e nel frattempo svolgere altre operazioni.

Il funzionamento interno della funzione CheckIO è molto semplice. Al pari delle altre funzioni, quando riceve il controllo si aspetta che nel registro A1 sia presente l'indirizzo della richiesta di I/O che dovrebbe prima o poi essere soddisfatta. La prima operazione che compie è controllare lo stato del flag IOF_QUICK: se risulta impostato significa che la richiesta (sia che riguardi un comando normale, sia che riguardi un comando immediato) è già stata elaborata e non è stata accodata alla reply port del task. In questo caso CheckIO memorizza l'indirizzo della richiesta in D0 e restituisce il controllo.

Se invece il flag IOF_QUICK risulta azzerato, la funzione procede a verificare il contenuto del parametro In_Type nella richiesta di I/O: se contiene la costante NT_REPLYMSG significa che il dispositivo ha restituito la richiesta al mittente, e quindi CheckIO memorizza l'indirizzo della richiesta nel registro D0. La richiesta si trova ora accodata alla reply port del task, il quale può accedervi anche subito, senza attendere che giunga alla sommità della coda, e può rimuoverla dalla coda tramite la funzione Remove. Se invece la verifica non ha successo, la funzione esce restituendo 0 nel registro D0. Come si può notare, CheckIO compie solo una verifica e non entra mai in attesa.

Le funzioni AddDevice e RemDevice

Questa sezione approfondisce quanto è stato esposto nel primo volume sulle funzioni AddDevice e RemDevice della libreria Exec. Prima di addentrarci a illustrarle premettiamo subito che nella gestione dei 12 dispositivi predefiniti dell'Amiga, un task non ha praticamente mai bisogno di chiamare AddDevice e RemDevice (anche se c'è un'eccezione per questa seconda funzione), in quanto è il sistema stesso a utilizzarle per lui. Giocano invece un ruolo importante quando un task deve interagire con un dispositivo che non rientra tra i 12 previsti dall'Amiga. In questo caso il dispositivo non è standard, e la funzione OpenDevice non è sufficiente per aprirlo. Quindi, ci soffermiamo su queste due funzioni solo per imparare a usarle nel caso che il dispositivo da aprire non sia standard.

AddDevice

AddDevice è una funzione della libreria Exec che rende disponibile il dispositivo indicato (si ricordi che con la parola disponibile si intende che il dispositivo si trova in memoria pronto per essere usato). L'argomento da passare a questa funzione è l'indirizzo della struttura Device del dispositivo. La sua funzione principale è quella di aggiungere alla lista di sistema DeviceList la struttura Device di un nuovo dispositivo, e si aspetta quindi che il dispositivo si trovi già in memoria (cioè, sia già stato allocato). Una volta che il suo nome è entrato nella DeviceList, qualsiasi task può servirsene per individuarlo. Al momento dell'accensione il sistema provvede ad aggiungere nella DeviceList

i dispositivi standard Audio, Input, Console, Gameport, Keyboard, Timer e TrackDisk, tutti residenti su ROM. Un dispositivo rimane nella lista fino a quando non viene rimosso esplicitamente tramite la funzione RemDevice. Si ricordi che per i 12 dispositivi predefiniti dell'Amiga, è sufficiente chiamare la funzione OpenDevice, la quale provvede automaticamente a rendere disponibile il dispositivo indicato.

RemDevice

RemDevice è una funzione della libreria Exec disalloca il dispositivo indicato. L'argomento da passare a questa funzione è l'indirizzo della struttura Device del dispositivo. La sua funzione principale è rimuovere dalla lista di sistema DeviceList il nodo corrispondente alla struttura Device indicata, e liberare completamente la memoria occupata dal dispositivo e dalle sue strutture (ovviamente, RemDevice può essere impiegata solo con i dispositivi residenti su disco, cioè quelli che vengono caricati in RAM, pena il crash del sistema). Per compiere questa operazione RemDevice chiama anche la routine Expunge del dispositivo, la quale provvede a eseguire le operazioni che assicurano una corretta chiusura del dispositivo.

I task possono usare RemDevice anche con i dispositivi predefiniti dell'Amiga. Se per esempio, un task ha aperto un dispositivo predefinito residente su disco, e dopo averlo usato lo chiude con la funzione CloseDevice, il dispositivo rimane sempre disponibile in memoria, anche se nessun altro task lo sta usando. Se la memoria RAM a disposizione è poca e ne occorre dell'altra, il task può decidere di eliminare il dispositivo dalla memoria; per farlo chiama la funzione RemDevice indicando l'indirizzo della struttura Device come argomento. Con questa operazione il dispositivo viene completamente eliminato, e la sua struttura Device non compare più all'interno della lista di sistema DeviceList.

L'esatto risultato ottenuto da RemDevice dipende dallo stato del dispositivo. Tuttavia, una regola è certa: RemDevice non potrà portare a termine suo compito fino a quando tutte le unità del dispositivo aperte con OpenDevice non sono state chiuse con CloseDevice. Si ricordi che ogni chiamata a OpenDevice e a CloseDevice rispettivamente apre e chiude un'unità di un dispositivo. In una chiamata alla funzione OpenDevice, la particolare unità che si intende aprire viene specificata come uno degli argomenti della chiamata alla funzione. Nella chiamata alla funzione CloseDevice, la particolare unità che si intende chiudere viene indicata implicitamente dal parametro io_Unit della struttura per la richiesta di I/O. Quindi, sebbene un dispositivo sia sempre aperto quando una qualsiasi delle sue unità è aperta, non si può ritenerlo completamente chiuso almeno fino a quando non lo sono tutte le sue unità.

Se un task chiama la funzione RemDevice per rimuovere un dispositivo quando una delle sue unità risulta ancora aperta, il sistema non elimina immediatamente il dispositivo. bensì inizializza il parametro lib_Flags a LIB_DELEXP nella struttura Device del dispositivo per indicare che l'eliminazione è stata prorogata. Per sapere se il dispositivo è completamente

inutilizzato, il sistema accede al parametro lib_OpenCnt della relativa struttura Device. Come si ricorderà, se questo parametro risulta azzerato, significa che nessun task nel sistema lo detiene aperto. In questo caso, il sistema può procedere a disallocarlo.

IMPIEGO DELLE FUNZIONI DI SUPPORTO ALLA LIBRERIA EXEC

In linguaggio C i compilatori hanno sempre bisogno di sapere che tipo di risultato restituiscono le funzioni, al fine di controllare se sono compatibili con le variabili alle quali i programmatori le assegnano. Per questa ragione, generalmente i programmatori eseguono operazioni di cast al fine d'indicare al compilatore i tipi di dato che deve associare ai valori restituiti dalle funzioni. Quando le funzioni restituiscono valori da 32 bit, non si verificano errori purché la compilazione venga eseguita facendo in modo che gli interi (tipo int) siano considerati da 32 bit. Se il programmatore non effettua operazioni di cast ed effettua la compilazione lasciando che gli interi vengano considerati da 16 bit, il codice in linguaggio macchina che ottiene non è sicuramente quello che si aspettava. In particolare, quello che accade in questo secondo caso è che le funzioni restituiscono valori da 32 bit, ma i codici generati dal compilatore li troncano a 16 bit per poi rieffettuarne il cast a 32 bit al fine di uniformarsi ai tipi di variabili indicate nel sorgente (che abbiamo supposto da 32 bit).

Per evitare il problema si possono scegliere due strade: effettuare il cast ogni volta che si chiama una funzione, o dichiararne il tipo in calce al sorgente. Ovviamente, la seconda soluzione è quella che permette di redigere sorgenti più ordinati e più brevi, in quanto la dichiarazione è globale. Nel caso delle funzioni di supporto alla libreria Exec che stiamo per descrivere, consigliamo di dichiarare esplicitamente nei propri sorgenti il tipo di variabili che restituiscono.

Le dichiarazioni per le funzioni CreatePort, CreateStdIO e CreateTask sono le seguenti:

```
struct MsgPort *CreatePort();
struct IOStdReq *CreateStdIO();
struct Task *CreateTask();
...
```

Non abbiamo incluso CreateExtIO in quanto questa funzione viene impiegata per allocare e inizializzare strutture di I/O non standard, che sono ovviamente diverse fra loro. A seconda del tipo di struttura non standard che il task deve allocare, i programmatori devono inserire la dichiarazione di tipo della funzione indicando quel tipo di struttura.

Per le altre funzioni di supporto alla libreria Exec non sono necessarie dichiarazioni di tipo in quanto non vengono restituiti valori significativi.

CreateExtIO

Sintassi di chiamata della funzione

iORequest = CreateExtIO (iOReplyPort, size)

Scopo della funzione

Questa funzione alloca e inizializza una struttura di tipo IOExtReq, memorizzando nel parametro iORequest->io_Message.mn_ReplyPort l'indirizzo della reply port indicato dall'argomento iOReplyPort. Una struttura IOExtReq è una struttura di I/O estesa non standard prevista da un particolare dispositivo; il numero di byte che occupa in memoria varia da dispositivo a dispositivo, e dev'essere specificato nel parametro size (di tipo LONG).

CreateExtIO restituisce l'indirizzo della struttura di I/O creata, che il task utilizzerà per inviare le proprie richieste di I/O al dispositivo. Si noti che l'indirizzo restituito dalla funzione è (dal punto di vista del C) di tipo struct IORequest, ma con una semplice conversione di tipo (cast) può diventare l'indirizzo di qualsiasi tipo di struttura estesa non standard. Viene restituito il tipo struct IORequest in quanto questa struttura è il primo elemento di qualsiasi struttura di I/O, standard, non standard ed estesa.

Argomenti della funzione

iOReplyPort — Deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Prima di chiamare la funzione, il task deve aver già provveduto ad allocare la reply port. Si ricordi che la reply port che viene indicata costituisce il mittente di tutte le richieste che vengono inviate tramite la struttura di I/O allocata e inizializzata da CreateExtIO.

size — Deve contenere la grandezza della struttura di I/O estesa, espressa in byte. Questo argomento dev'essere di tipo LONG.

Discussione

Due funzioni di supporto alla libreria Exec si occupano delle strutture di I/O estese non standard: CreateExtIO e DeleteExtIO. Si può utilizzare la funzione CreateExtIO per allocare e inizializzare nei propri task le strutture di I/O estese non standard richieste dal particolare dispositivo che si intende utilizzare. I dispositivi Serial e Parallel costituiscono due validi esempi di dispositivi che impiegano strutture di I/O estese non standard. CreateExtIO svolge le seguenti operazioni. Alloca in memoria la quantità di byte indicata dall'argomento size. Inizializza il parametro iORequest–> io_Message.mn_Node.ln_Type con la costante NT_MESSAGE, il parametro iORequest–>io_Message.mn_Node.ln_Pri a zero, e il parametro iORequest–>io_Message.mn_ReplyPort con l'indirizzo della reply port indicata dall'argomento iOReplyPort. L'azzeramento del parametro ln_Pri serve per indicare al sistema che la richiesta, quando viene inviata alla message port, dev'essere sempre inserita al fondo della coda. La funzione utilizza l'argomento size per allocare lo spazio di memoria necessario, e poi lo memorizza nel parametro mn_Length della sotto-struttura Message che intesta la struttura di I/O appena creata, parametro che verrà poi consultato da DeleteExtIO. Per questa ragione, se si usano CreateExtIO e DeleteExtIO il parametro mn_Length (attenzione, non io_Length) non dev'essere mai alterato.

Dato che la struttura IORequest è il primo elemento di ogni struttura di I/O, un puntatore che la individua è anche un puntatore alla struttura di I/O estesa. Ogni dispositivo che non utilizza la struttura IOStdReq (l'estensione standard della struttura IORequest) richiede una propria struttura di I/O estesa non standard; la sua grandezza dipende dalle necessità di dati del dispositivo. Tutte queste strutture di I/O specifiche per ogni dispositivo sono definite nei file INCLUDE (si veda la Tavola 1.2 a pagina 32). Un task può utilizzare l'operatore sizeof del linguaggio C per conoscere il numero dei byte richiesti da una struttura di I/O estesa e indicarla come argomento della funzione (si ricordi che l'operatore sizeof restituisce un intero, e che quindi occorre effettuare la conversione di tipo in LONG).

CreatePort

Sintassi di chiamata della funzione

msgPort = CreatePort (msgPortName, msgPort_Priority)

Scopo della funzione

Questa funzione alloca, inizializza e dichiara una struttura MsgPort assegnandole il nome e la priorità indicati come argomenti (si ricordi che una struttura MsgPort costituisce una message port). CreatePort assegna alla message port un bit di segnale, in modo che un task in attesa possa riottenere il controllo. Inoltre, se il primo argomento è diverso da 0, la funzione aggiunge la message port alla lista di sistema delle message port, utilizzando l'argomento msgPort_Priority per stabilirne la posizione nella lista.

L'argomento msgPortName costituisce il nome proprio della message port, e fornisce un modo agli altri task per individuarla tramite la funzione FindPort. Comunque, nella gestione dei dispositivi non è assolutamente necessario che le message port vengano allocate come pubbliche. La funzione CreatePort restituisce l'indirizzo della struttura MsgPort della message port appena creata.

Argomenti della funzione

msgPortName Questo argomento è costituito da una stringa di caratteri contenente il nome proprio che si desidera assegnare alla message port; il valore di questo argomento viene assegnato al parametro ln_Name della sotto-struttura Node contenuta nella struttura MsgPort.

msgPort_Priority Questo argomento indica la priorità (il cui range di validità è da −128 a +127) che si vuole assegnare alla message port, e determina la posizione che essa assumerà nella lista di sistema PortList; il suo contenuto viene memorizzato nel parametro ln_Pri della sotto-struttura Node contenuta nella struttura MsgPort. Dev'essere di tipo LONG.

Discussione

Due funzioni di supporto alla libreria Exec si occupano delle message port: CreatePort e DeletePort. Si utilizza CreatePort per creare, allocare e inizializzare le message port per i propri task. Queste message port possono essere utilizzate per accodare qualsiasi messaggio, prescindendo dalla sua origine. Nella programmazione dei dispositivi, queste porte agiscono come reply port per le richieste di I/O restituite dai dispositivi.

CreatePort svolge le seguenti mansioni. Alloca l'area di memoria necessaria per contenere la struttura MsgPort. Alloca un bit di segnale nella struttura Task del task, e memorizza il numero di questo bit nel parametro

mp_SigBit della struttura MsgPort che crea. Memorizza nel parametro mp_SigTask l'indirizzo della struttura Task del task, in modo che quando giunge un messaggio, il sistema sia in grado d'inviare un segnale al task. Imposta i parametri ln_Name, ln_Pri, ln_Type della sotto-struttura Node. Inserisce la message port nella lista di sistema PortList purché il task abbia indicato come primo argomento un indirizzo non nullo.

CreateStdIO

Sintassi di chiamata della funzione

iOStdReq = CreateStdIO (iOReplyPort)

Scopo della funzione

Questa funzione alloca e inizializza una struttura di tipo IOStdReq. memorizzando nel parametro iOStdReq->io_Message.mn_ReplyPort l'indirizzo della reply port indicato dall'argomento iOReplyPort. CreateStdIO imposta a 0 il parametro ln_Pri, contenuto nella sotto-struttura Node della struttura IOStdReq, per indicare che la richiesta di I/O dev'essere sempre collocata in fondo alla coda della message port.

CreateStdIO restituisce l'indirizzo della struttura IOStdReq appena creata. Un task può utilizzare questa struttura per inviare un comando a qualsiasi unità di un dispositivo che ne prevede l'impiego.

Argomenti della funzione

iOReplyPort — Deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task.

Discussione

Due funzioni di supporto alla libreria Exec si occupano delle strutture di I/O estese standard: CreateStdIO e DeleteStdIO. Si può utilizzare la funzione CreateStdIO per allocare e inizializzare le strutture di I/O estese standard richieste dal dispositivo che s'intende utilizzare. CreateStdIO svolge le seguenti operazioni. Alloca in memoria la quantità di byte sufficienti per

contenere la struttura IOStdReq. Inizializza il parametro iORequest->io_Message.mn_Node.ln_Type con la costante NT_MESSAGE, il parametro iORequest->io_Message.mn_Node.ln_Pri a zero, e il parametro iORequest->io_Message.mn_ReplyPort con l'indirizzo della reply port indicata dall'argomento iOReplyPort.

CreateTask

Sintassi di chiamata della funzione

taskCB = CreateTask (taskName, task_priority, taskEntryPoint, task_stack_size)

Scopo della funzione

CreateTask alloca in memoria una struttura Task, lo stack a essa associato, e aggiunge nel sistema il task, mandandolo così in esecuzione. La funzione inizializza anche alcuni parametri della struttura Task. Imposta la priorità del task con il valore specificato dall'argomento task_priority (la priorità viene memorizzata nel parametro ln_Pri della sotto-struttura Node appartenente alla struttura Task). Prende atto dell'esatta locazione RAM dalla quale deve iniziare l'elaborazione del task, inizializza i parametri di controllo dello stack (tc_SPReg, tc_SPUpper e tc_SPLower), e tramite la funzione AddTask aggiunge il task alla lista di sistema TaskReady nella posizione indicata dall'argomento task_priority. Si ricordi che il sistema manda periodicamente in esecuzione tutti i task presenti in questa lista.

Una volta compiute queste operazioni, la funzione CreateTask restituisce l'indirizzo della struttura Task che ha creato e che verrà utilizzata per controllare il task.

Argomenti della funzione

taskName — Deve indicare la stringa di caratteri che costituisce il nome del task; l'indirizzo di questa stringa viene assegnato al parametro ln_Name della sotto-struttura Node appartenente alla struttura Task.

task_priority — Indica la priorità (il cui valore può variare tra −128 e +127) del task da mandare in esecuzione. Questo argomento dev'essere di tipo LONG.

taskEntryPoint Deve indicare l'indirizzo della locazione di memoria dalla quale avrà inizio l'esecuzione del task; viene utilizzato come parametro (initPC) della funzione AddTask chiamata all'interno della funzione CreateTask.

task_stack_size Deve contenere la dimensione dello stack che CreateTask alloca e assegna al task (si tratta di una LONG); viene utilizzato dalla funzione CreateTask per stabilire il valore dei parametri di controllo dello stack.

Discussione

Due funzioni di supporto alla libreria Exec vengono utilizzate per gestire i task: CreateTask e DeleteTask. Queste funzioni attivano e concludono l'esecuzione di un task, ne controllano l'allocazione e la disallocazione in RAM, i segnali e altre operazioni di conteggio richieste per tenere sotto controllo le risorse utilizzate dal task.

L'utilizzo delle funzioni CreateTask e DeleteTask non è ristretto ai task di gestione dei dispositivi: in realtà è possibile utilizzarle per creare ed eliminare qualsiasi task. Queste funzioni, inoltre, possono essere destinate anche a impieghi particolari; si consulti l'appendice, che ne presenta le definizioni in linguaggio C.

DeleteExtIO

Sintassi di chiamata della funzione

DeleteExtIO (iOExtReq)

Scopo della funzione

Questa funzione disalloca la memoria occupata da una struttura di I/O estesa non standard allocata tramite la funzione CreateExtIO.

Argomenti della funzione

iOExtReq Deve contenere l'indirizzo della struttura di I/O estesa non standard da eliminare. Deve trattarsi del puntatore originariamente restituito dalla funzione CreateExtIO.

Discussione

Due funzioni di supporto alla libreria Exec si occupano delle strutture di I/O estese non standard: CreateExtIO e DeleteExtIO. DeleteExtIO disalloca tutta la memoria assegnata a una struttura di I/O estesa. Per conoscere le dimensioni dell'area RAM da liberare, la funzione accede al parametro mn_Length della sotto-struttura Message, nel quale si aspetta di trovare la dimensione in byte della struttura di I/O. Se la struttura è stata creata da CreateExtIO questo parametro è già pronto. Si può utilizzare la funzione DeleteExtIO per eliminare qualsiasi struttura di I/O estesa non standard nel sistema. Si tenga presente che DeleteExtIO non disalloca la reply port associata alla struttura di I/O indicata.

DeletePort

Sintassi di chiamata della funzione

DeletePort (msgPort)

Scopo della funzione

Questa funzione rimuove dalla lista di sistema PortList la struttura MsgPort indicata come argomento se risulta pubblica, e la disalloca liberando la memoria da essa occupata. Infine, DeletePort chiama la funzione FreeSignal dell'Exec per rendere nuovamente libero il numero del bit di segnale assegnato alla message port dalla funzione CreatePort.

Argomenti della funzione

msgPort Deve contenere l'indirizzo della struttura MsgPort che dev'essere eliminata; è costituito normalmente dall'indirizzo restituito dalla funzione CreatePort.

Discussione

La funzione DeletePort viene utilizzata per liberare la RAM originariamente allocata dalla funzione CreatePort per la struttura MsgPort. Si noti che qualsiasi task può chiamare la funzione DeletePort per eliminare una message port dal sistema, e che la message port da eliminare non deve necessariamente aver avuto origine da una chiamata a CreatePort. La funzione DeletePort si può anche usare per le message port che non sono state inserite nella lista di sistema PortList, cioè che non sono state create come pubbliche.

DeleteStdIO

Sintassi di chiamata della funzione

DeleteStdIO (iOStdReq)

Scopo della funzione

Questa funzione libera la memoria occupata dalla struttura IOStdReq indicata come argomento. Questa struttura viene generalmente creata tramite la funzione CreateStdIO.

Argomenti della funzione

iOStdReq Questo parametro deve contenere l'indirizzo della struttura IOStdReq da eliminare.

Discussione

Si utilizza la funzione DeleteStdIO per liberare la memoria allocata per una struttura di tipo IOStdReq. Si noti che qualsiasi task può utilizzare DeleteStdIO per eliminare una struttura IOStdReq dal sistema. La struttura da eliminare non deve necessariamente aver avuto origine da una chiamata alla funzione CreateStdIO.

DeleteTask

Sintassi di chiamata della funzione

DeleteTask (taskCB)

Scopo della funzione

Questa funzione libera la memoria RAM occupata da un task, dalla sua relativa struttura Task e dal corrispondente user stack. Il task cessa di esistere, e la sua struttura Task viene rimossa dalla lista di sistema TaskWait o TaskReady tramite la funzione RemTask della libreria Exec.

Argomenti della funzione

taskCB Questo argomento è l'indirizzo della struttura Task relativa al task da rimuovere.

Discussione

CreateTask e DeleteTask devono essere utilizzate in coppia; agendo in questo modo il programmatore ottiene infatti un notevole vantaggio. Allo stesso modo in cui si utilizza la funzione CreateTask per creare la struttura Task si utilizza la funzione DeleteTask per eliminarla.

NewList

Sintassi di chiamata della funzione

NewList (list)

Scopo della funzione

Questa funzione inizializza una nuova lista di sistema impiegando la macro NEWLIST in linguaggio Assembly.

Argomenti della funzione

list	Deve contenere l'indirizzo della struttura List che controllerà la nuova lista nel sistema.

Discussione

La funzione NewList impiega la macro NEWLIST in linguaggio Assembly per inizializzare i parametri della struttura List. Quando il task riottiene il controllo, può aggiungere o rimuovere nodi dalla lista definendo appropriate strutture Node.

Il dispositivo Audio

Introduzione

Questo capitolo analizza il dispositivo Audio, il più complesso del sistema Amiga. Dal momento che deve permettere a più task di condividere contemporaneamente i suoi canali audio (le sue quattro unità), il dispositivo Audio possiede un meccanismo di gestione che provvede a ripartire i canali audio, ad autorizzare l'accesso per i task e a tenere informati altri task sullo stato dei quattro canali. Grazie alle capacità multitasking dell'Amiga, il dispositivo Audio permette inoltre il funzionamento nel modo double-buffered, che ne complica ulteriormente la programmazione. Se ne parlerà diffusamente descrivendo il comando CMD_WRITE.

Per interagire con il dispositivo Audio vengono utilizzate quattro funzioni: AbortIO, BeginIO, OpenDevice, e CloseDevice. Le funzioni DoIO e SendIO, invece, non dovrebbero mai essere impiegate; al loro posto si consiglia di usare sempre la funzione BeginIO, che, ricordiamo, può inviare le richieste in modo sincrono o asincrono a seconda delle condizioni in cui si trova il sistema. La ragione principale per cui queste funzioni non devono essere usate con il dispositivo Audio è che azzerano tutti i bit del parametro io_Flags della struttura IOAudio, e alcuni comandi del dispositivo Audio considerano questi bit come parametri. Inoltre l'impiego di BeginIO delega al dispositivo la scelta fra l'I/O sincrono e asincrono, rendendo un po' più semplice la gestione di un dispositivo estremamente complesso.

Il sistema hardware del dispositivo Audio

La configurazione hardware del dispositivo Audio è illustrata nella Figura 3.1 (nella pagina successiva). Ciascun canale audio possiede un convertitore digitale-analogico a 8 bit pilotato da un canale DMA (Direct Memory Access – accesso diretto alla memoria), il quale provvede a leggere nella chip RAM i valori dell'ampiezza che riprodotti in sequenza generano il segnale audio. Perché si possa generare un suono da un canale audio, occorre allocare nella chip RAM un array e definire al suo interno un ciclo campionato della forma d'onda da riprodurre (campionare un segnale significa rilevare la sua ampiezza a intervalli di tempo regolari). Ogni elemento dell'array è un *dato campione della forma d'onda*, cioè l'ampiezza della forma d'onda corrispondente a un particolare punto del suo periodo, e occupa un byte. Per risparmiare chip RAM, il task definisce in memoria un solo ciclo campionato della forma d'onda, che il canale audio provvede poi a riprodurre periodicamente per creare il segnale audio; ovviamente, questo è possibile solo se il suono che s'intende generare è periodico. Se al contrario, il suono non è periodico, il task deve costruire una forma d'onda campionata che descriva il suono per tutta la sua durata, e richiedere al canale audio di riprodurla una sola volta. L'array della forma d'onda può contenere da un minimo di due dati campione a un massimo di

131.072, e il numero di dati presenti dev'essere sempre pari, in quanto i canali DMA prelevano due dati alla volta. Inoltre, l'inizio dell'array in memoria deve corrispondere a un indirizzo pari, in modo che le coppie di elementi in esso presenti siano allineate con le word.

Ogni canale DMA può prelevare due dati campione per ogni scansione orizzontale di una linea di schermo effettuata dal pennello elettronico (64 ms nel sistema PAL). Si tratta della massima velocità di riproduzione dei dati campione, e come vedremo è un dato di cui occorre tener conto quando si richiede al dispositivo Audio di generare suoni.

Ogni canale dispone di un proprio controllo sul volume. Il volume del suono prodotto da un canale audio può variare da 0 a 64; si tratta di una scala arbitraria, nella quale ogni valore corrisponde a un particolare livello di decibel. Per un volume regolato al livello 64, considerando valori dei dati campione oscillanti fra i limiti +127 e −128 (estremi compresi), le massime tensioni elettriche disponibili in output dall'Amiga verso un amplificatore audio esterno sono comprese fra 0,4 volt e −0,4 volt; in questo caso si ottengono 0,0 decibel. Se invece il volume diventa 1, quasi il livello minimo, si ottiene un'attenuazione del segnale pari a −36,1 decibel.

La tonalità del suono ottenuto tramite la riproduzione successiva dei dati campione dipende dalla rapidità con la quale questi dati vengono riprodotti. Il tempo che intercorre fra la riproduzione di un dato campione e il successivo

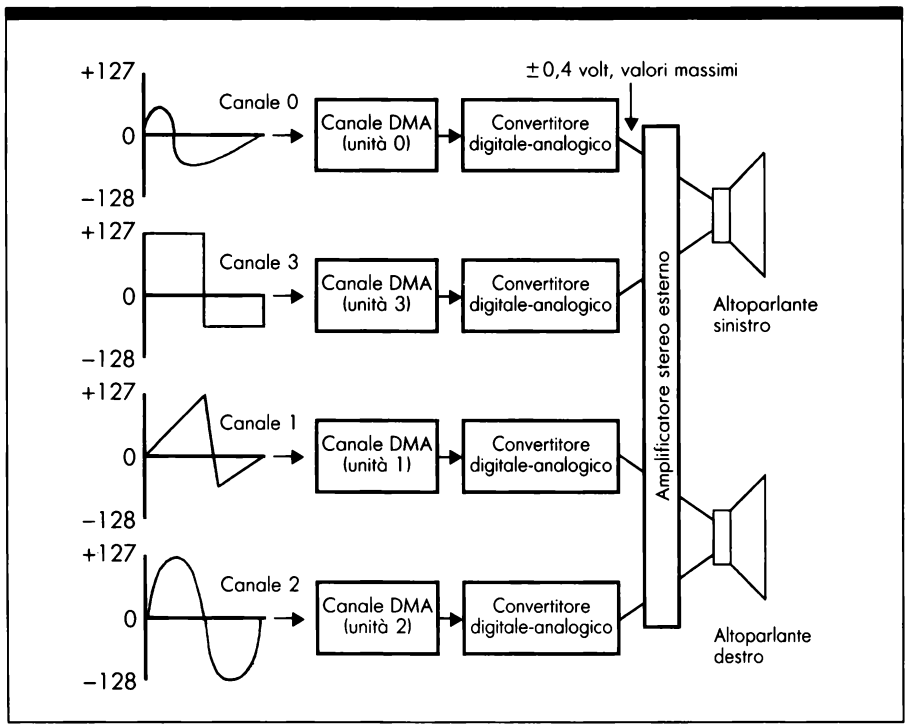


Figura 3.1: *Configurazione hardware del dispositivo Audio*

viene definito *periodo di campionamento*, ed è espresso in tick di sistema (un tick di sistema equivale a due cicli di clock, cioè a 279,365 ns nei sistemi americani e 281,932 ns nei sistemi europei). Non lo si confonda con il periodo della forma d'onda (l'inverso della frequenza), cioè il numero di cicli al secondo. Fra le due quantità esiste comunque una precisa relazione. Infatti, dal momento che i dati campione riproducono il ciclo di una forma d'onda periodica, conoscendo la loro quantità, la rapidità con cui vengono riprodotti, e la durata di un tick di sistema, si risale facilmente al periodo del suono. La formula è la seguente:

Periodo del suono = (periodo di campionamento) * (numero di dati campione) * (tick di sistema)

Calcolando l'inverso del periodo si risale alla frequenza. Supponiamo che la macchina di cui disponiamo sia europea, cioè che abbia un clock di 7,0939 MHz (tick di sistema pari a 281,932 ns). Con un periodo di campionamento pari a 504 tick, e con un array della forma d'onda composto da 16 dati campione/ciclo, si ottiene un suono di frequenza molto prossima a 440 Hz (439,85), la frequenza della nota fondamentale "La". Se la forma d'onda fosse stata definita da 32 dati campione/ciclo, per ottenere 440 Hz avremmo dovuto impostare un periodo di campionamento pari a 252 tick.

Con il dispositivo Audio le richieste di I/O devono essere formulate tramite la struttura non standard IOAudio. Al suo interno, il parametro ioa_Period serve per indicare il periodo di campionamento della forma d'onda da riprodurre. espresso naturalmente in forma intera. Il fatto che non si possa esprimere il periodo di campionamento con maggior precisione non deve comunque spaventare: per esempio, la differenza di 0,149 Hz fra la nota fondamentale La (periodo di campionamento teorico pari a 503,828 tick) e il suono che si ottiene indicando un periodo di campionamento pari a 504 tick è del tutto impercettibile all'orecchio umano. Quest'esempio vuole sottolineare un'importante differenza fra i sistemi americani ed europei che si riflette pesantemente sul funzionamento del dispositivo Audio e di cui occorre tenere conto nello sviluppo di software musicale. Volendo ottenere la nota fondamentale La con un Amiga americano, infatti, occorre indicare un periodo di campionamento pari a 508 tick. Se prendiamo un programma musicale nato esclusivamente per il mercato statunitense e lo mandiamo in esecuzione su un Amiga europeo, la nota che otteniamo si discosta dal La di ben 3,61 Hz, uno scarto perfettamente udibile.

Ogni canale audio possiede un registro di periodo, il cui valore viene utilizzato per effettuare un conto alla rovescia espresso in tick di sistema; ogni volta che il registro di periodo assume il valore zero, viene riprodotto un nuovo dato campione dell'array che definisce la forma d'onda. Quindi, il registro di periodo deve contenere il valore del periodo di campionamento con il quale si desidera sia generato il suono. Il valore massimo è 65.535 tick, mentre il valore minimo è 124. Un valore basso corrisponde a un'alta frequenza mentre un valore elevato corrisponde a un suono di bassa frequenza. Se il valore impostato per un canale fosse inferiore a 124, il relativo DMA audio non farebbe in tempo a prelevare la successiva coppia di dati campione dalla memoria quando il

registro di periodo si azzera, e quindi restituirebbe la coppia prelevata precedentemente.

Dato che l'Amiga è un sistema multitasking, diversi task possono competere tra loro per appropriarsi dei quattro canali audio disponibili. In qualsiasi momento un task può richiedere uno o più canali in uso da altri task. È il dispositivo Audio che decide come ripartire i quattro canali fra i task che, istante per istante, li richiedono.

I canali del dispositivo Audio sono caratterizzati da due stati: possono risultare *assegnati* a un task, oppure *liberi*. Quando parleremo dell'azione intrapresa da un task per farsi assegnare i canali di cui ha bisogno, diremo che il task "richiede l'accesso" ai canali.

Dal momento che la ripartizione multitasking dei canali non è affatto semplice, è di assoluta competenza del dispositivo: i task non possono intromettersi. Tuttavia, è compito del task segnalare l'urgenza con la quale ha bisogno dei canali audio. I meccanismi a disposizione del task per indicare le proprie esigenze sono due: l'*array di combinazioni dei canali* e la *priorità d'assegnazione dei canali*.

L'array di combinazioni dei canali

L'array di combinazioni dei canali è lo strumento tramite il quale il task indica al dispositivo di quali canali audio ha bisogno. Il task inserisce in questo array un certo numero di alternative sperando che almeno una venga soddisfatta (ovviamente le combinazioni devono essere tutte diverse).

La Tavola 3.1 (nella pagina successiva) illustra le possibili combinazioni. Dato che esistono in tutto quattro canali, escludendo la combinazione nulla si possono indicare 15 combinazioni, comprendenti tutte le possibili ripartizioni sia singole (un canale solo) sia multiple (più canali). Ogni combinazione occupa un byte dell'array, ma solo i bit del nibble meno significativo vengono usati per indicare le unità che si desiderano. Ognuno dei quattro bit corrisponde a un particolare canale: i bit a 1 indicano al dispositivo quali sono i canali richiesti. L'unità 0 è rappresentata dal bit meno significativo, mentre l'unità 3 è rappresentata dal bit più significativo del nibble. L'unità 0 e l'unità 3 sono collegate all'altoparlante sinistro, l'unità 1 e l'unità 2 all'altoparlante destro, come si vede nella Figura 3.1 (a pagina 80).

Vediamo ora due esempi di array di combinazioni dei canali. Se il task deve riprodurre suoni attraverso l'altoparlante sinistro, e non ha preferenze sul canale audio, ha più probabilità di ottenere l'accesso se indica nell'array le 12 combinazioni che comprendono l'assegnazione dell'altoparlante sinistro (1, 3, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15); in questo caso l'array è composto da 12 byte. Più alto è il numero di valori distinti (combinazioni) che un task specifica nell'array di combinazioni dei canali. maggiore è la probabilità che i canali richiesti (in questo caso almeno uno dei canali collegati all'altoparlante sinistro) gli vengano assegnati.

Se però il task ha necessità di restringere la varietà delle combinazioni, deve indicare nell'array un minor numero di valori. Per esempio, se il task deve

Tavola 3.1: *Possibili combinazioni dei canali audio*

Canale 3	Canale 2	Canale 1	Canale 0	Numero della combinazione	Altoparlante sinistro	Altoparlante destro	Suono stereofonico	Modulazione in frequenza o in ampiezza
0	0	0	0	0	–	–	–	–
0	0	0	1	1	✓	–	–	–
0	0	1	0	2	–	✓	–	–
0	0	1	1	3	✓	✓	✓	1 e 0 uniti
0	1	0	0	4	–	✓	–	–
0	1	0	1	5	✓	✓	✓	–
0	1	1	0	6	–	✓	–	2 e 1 uniti
0	1	1	1	7	✓	✓	✓	–
1	0	0	0	8	✓	–	–	–
1	0	0	1	9	✓	–	–	–
1	0	1	0	10	✓	✓	✓	–
1	0	1	1	11	✓	✓	✓	–
1	1	0	0	12	✓	✓	✓	3 e 2 uniti
1	1	0	1	13	✓	✓	✓	–
1	1	1	0	14	✓	✓	✓	–
1	1	1	1	15	✓	✓	✓	–

riprodurre suoni *solo* dall'altoparlante sinistro e non vuole che gli vengano assegnati il canale 0 e il canale 3 contemporaneamente, deve specificare un array di combinazioni dei canali composto da due byte, uno contenente il valore 1 e l'altro contenente il valore 8. Nessuna di queste due combinazioni produce suoni dall'altoparlante destro, né assegna al task entrambi i canali dell'altoparlante sinistro.

Nelle strutture di I/O l'array di combinazioni dei canali viene indicato memorizzandone l'indirizzo nel puntatore ioa_Data, e il numero di combinazioni di cui è composto (cioè il numero di byte che occupa) nel parametro ioa_Length (possibili valori 0-15) della struttura IOAudio. Nell'ultimo esempio illustrato, ioa_Length dovrebbe contenere il valore 2. Una volta che i parametri dell'array (indirizzo e dimensione) sono stati inseriti nella struttura di I/O, questa può essere indicata come argomento della funzione OpenDevice (se si desidera assegnare i canali al momento dell'apertura del dispositivo) o può costituire la richiesta di I/O del comando ADCMD_ALLOCATE. Le routine interne del dispositivo Audio elaborano la richiesta valutando ogni combinazione indicata dall'array parallelamente allo stato d'assegnazione dei canali. Si noti che OpenDevice, oltre ad aprire il dispositivo, tenta di assegnare le combinazioni di canali indicate dal task nell'array solo se rileva che il parametro ioa_Length della struttura di I/O è diverso da zero; questo modo di richiedere canali equivale a inviare il comando ADCMD_ALLOCATE. In entrambi i casi, l'array di combinazioni viene sempre impiegato nel modo seguente:

- il dispositivo preleva il primo elemento dell'array e verifica se i canali indicati possono essere assegnati. Se la risposta è affermativa, li assegna al task e provvede a copiare nel parametro io_Unit della struttura IOAudio l'elemento dell'array appena prelevato; quando il task ottiene la risposta alla sua richiesta, va a leggere il valore contenuto nel parametro io_Unit per rilevare quale combinazione tra quelle indicate ha avuto successo (si noti che per il dispositivo Audio il parametro io_Unit assume un significato completamente diverso da quello discusso nei precedenti capitoli; spiegheremo in seguito il motivo di quest'eccezione).

- Può capitare che il primo elemento dell'array non venga accettato; questo accade quando la combinazione di canali richiesta ha almeno un canale in comune con una delle combinazioni già assegnate. Il dispositivo procede allora ad analizzare gli elementi che seguono, fino a quando uno di essi non ha successo. La combinazione che ha successo viene in ogni caso memorizzata nel parametro io_Unit della struttura IOAudio.

- Se il dispositivo ha esaminato tutti gli elementi dell'array e non è riuscito ad assegnare nessuna delle combinazioni di canali richieste, tenta allora di sottrarre canali ad altri task. Accede al parametro ln_Pri della struttura di I/O (contenente la priorità d'assegnazione che il task ha definito per il suo array), e confronta la priorità indicata con le priorità

dei canali che risultano già assegnati. Quindi scandisce nuovamente l'array alla ricerca di una combinazione che eventualmente sottragga ad altri task canali con una priorità d'assegnazione più bassa. Si noti che il dispositivo non prende in considerazione assegnazioni "parziali". Facciamo un esempio: supponiamo di avere un task che ha accesso ai canali 0 e 1 con priorità 0, e un altro task che ha accesso ai canali 2 e 3 con priorità 20; se un terzo task cerca di ottenere l'accesso alla combinazione 6 (canale 1 e 2) con una priorità superiore a 0 ma inferiore a 20, la richiesta non viene soddisfatta fino a quando il task con priorità d'assegnazione pari a 20 non libera i suoi canali. In altre parole, non viene presa in considerazione l'assegnazione del solo canale 1, che potrebbe essere facilmente sottratto al primo task.

La priorità d'assegnazione dei canali

La priorità d'assegnazione dei canali rappresenta l'importanza di una richiesta per il task. Può variare fra +127 e −128, e dev'essere indicata dal task nel parametro ln_Pri della struttura IOAudio ogni volta che effettua una richiesta d'assegnazione dei canali, cioè invia il comando ADCMD_ALLOCATE (tramite la funzione BeginIO), o chiama la funzione OpenDevice con ioa_Length diverso da zero.

Un valore di −128 nel parametro ln_Pri indica che la richiesta ha priorità minima, mentre un valore di +127 indica che il task pretende i canali subito. Se l'altoparlante sinistro deve emettere un suono immediatamente, si deve impostare il parametro ln_Pri a +127 e creare un array di combinazioni dei canali con tutti i valori che ne prevedono l'assegnazione. Se invece il suono può essere emesso in un momento qualunque, conviene impostare il parametro ln_Pri con un valore molto più basso.

Se l'array contiene combinazioni che prevedono l'assegnazione di più canali contemporaneamente e nessuna ha avuto successo perché la priorità indicata è bassa, si può tentare di dividere la richiesta in diverse richieste di singoli canali, ognuna con la propria priorità e il proprio array di combinazioni. In questo modo l'assegnazione diventa più probabile.

Quando si invia il comando ADCMD_ALLOCATE, un altro modo per indicare al dispositivo che si desidera accedere immediatamente ai canali consiste nell'impostare il flag ADIOF_NOWAIT nel parametro io_Flags della richiesta di I/O. Questo flag fa in modo che il task ottenga subito la risposta al comando anche se nessuna delle combinazioni ha successo, per esempio quando in ogni combinazione esiste un canale detenuto da un altro task con la medesima priorità (in questo caso, ADCMD_ALLOCATE restituisce nel parametro io_Error il codice d'errore ADIOERR_ALLOCFAILED). L'unica eccezione a questo comportamento si verifica quando uno dei canali richiesti risulta protetto (con qualsiasi priorità): in questo caso, infatti, anche se è stato impostato il flag ADIOF_NOWAIT il dispositivo attende che il canale protetto venga liberato senza preoccuparsi di restituire al task la risposta. Lo stato del flag ADIOF_NOWAIT diventa ininfluente.

Se invece ADIOF_NOWAIT non è impostato, la risposta al comando ADCMD_ALLOCATE non perviene finché una delle combinazioni non ha successo, e se il task desidera eliminare il comando deve utilizzare la funzione AbortIO. In entrambi i casi, comunque, il task riottiene il controllo subito dopo l'esecuzione della funzione BeginIO e può controllare periodicamente con CheckIO se la risposta del dispositivo è arrivata. Quindi, se ADIOF_NOWAIT è impostato il comando viene eseguito in modo sincrono, e quando riottiene il controllo il task può già visitare la struttura di I/O per rilevare l'esito della richiesta. Se invece ADIOF_NOWAIT è azzerato, la richiesta viene gestita in modo asincrono: il controllo viene subito restituito al task e intanto il dispositivo cerca di soddisfare la richiesta (il task riceve la risposta nella sua reply port solo quando il dispositivo è riuscito a soddisfare la richiesta).

Per assegnare i canali, però, si può usare anche la funzione OpenDevice, la quale restituisce sempre la struttura di I/O senza aspettare che si liberino i canali (come se fosse stato impostato il flag ADIOF_NOWAIT). Anche in questo caso esiste però un'eccezione. Se infatti uno dei canali richiesti risulta protetto da un altro task con priorità minore di quella indicata nella richiesta, la funzione OpenDevice mette il task in attesa, e restituisce il controllo solo quando quel canale viene liberato. Questo "congelamento" del task rappresenta un caso unico per il dispositivo Audio, dal momento che in tutti gli altri casi le richieste che non sono esaudite subito vengono trattate in modo asincrono.

Protezione e sottrazione dei canali

Il comando ADCMD_LOCK permette a un task di attivare un sistema di protezione su uno o più canali audio in suo possesso. Questa esigenza può presentarsi se il task deve lavorare direttamente con i registri hardware del dispositivo audio e desidera che nessun altro task se ne appropri senza la sua autorizzazione. Infatti, se un task si appropriasse di un canale indicando un'elevata priorità d'assegnazione, mentre un altro ne sta impiegando direttamente i registri hardware, quest'ultimo non avrebbe modo di saperlo, e potrebbe compromettere il lavoro del primo task.

Se invece il task, dopo aver ricevuto il canale, invia il comando ADCMD_LOCK, una successiva richiesta a priorità più alta viene momentaneamente sospesa, e il dispositivo restituisce al task la risposta al comando ADCMD_LOCK con il codice d'errore ADIOERR_CHANNELSTOLEN nel parametro io_Error; quando arriva questa risposta, il task che sta operando sul canale protetto dovrebbe smettere d'impiegarlo al più presto e inoltrare il comando ADCMD_FREE, o chiamare la funzione CloseDevice (che comunque provvede a eseguire il comando ADCMD_FREE per tutti i canali indicati dalla chiave d'assegnazione). L'ultima parola, in ogni caso, spetta sempre al task che ha protetto i canali.

ADCMD_LOCK svolge quindi due funzioni: impedisce che un altro task con una richiesta d'assegnazione a priorità più alta si appropri dei canali, e avverte il task che uno o più dei suoi canali sono richiesti da un altro task. Per inviare questo comando, il task deve copiare nella struttura IOAudio del comando ADCMD_LOCK il parametro ioa_AllocKey che ha ottenuto assieme all'assegna-

zione dei canali, e impostare nel parametro io_Unit i bit relativi ai canali da proteggere.

Occorre evitare che la liberazione di un canale protetto dipenda dall'assegnazione di un altro canale, in quanto potrebbe instaurarsi una protezione perpetua: un task richiede l'accesso a un canale assegnato a un altro task, ma questo a sua volta potrebbe non dare l'autorizzazione a procedere fino a quando il primo task non ottiene l'accesso al canale.

Per evitare la perdita di un canale, un task può anche inizializzarne la priorità al massimo valore consentito (+127); anche questa è una protezione. con la differenza che il task non viene avvisato se altri task desiderano quel canale. Se l'esigenza è portare al massimo la priorità di un canale, si eviti di usare il comando ADCMD_LOCK (che in realtà non modifica la priorità), e s'impieghi invece il comando ADCMD_SETPREC.

La chiave d'assegnazione

Per alcuni dispositivi dell'Amiga, può essere richiesta la condivisione delle unità tra diversi task impostando nella struttura di I/O per l'apertura del dispositivo il flag IOF_SHARED. In particolare, questo è il modo in cui lavorano i dispositivi Serial e Parallel (nelle cui strutture di I/O il flag da impostare prende il nome di SERF_SHARED e PARF_SHARED). Se questo flag non è impostato, il dispositivo viene aperto nel modo di accesso esclusivo, che è il modo di default in cui si trovano la maggior parte dei dispositivi dell'Amiga al momento della loro apertura con OpenDevice. Inoltre, a parte il dispositivo Audio, tutti i dispositivi dell'Amiga permettono che i loro comandi agiscano sempre e solo su un'unità alla volta, cioè non permettono che uno stesso comando agisca su più unità contemporaneamente; per questo tipo di funzionamento, la struttura Unit che abbiamo illustrato nei precedenti capitoli è perfettamente idonea, dal momento che permette d'individuare ogni unità senza possibilità d'errore. Come vedremo meglio fra poco, non è altrettanto idonea per il dispositivo Audio.

Il dispositivo Audio si distingue dagli altri in quanto permette a uno stesso comando di agire contemporaneamente su più unità, o meglio su una combinazione di unità. Per funzionare in questo modo, le richieste di I/O che i task inviano al dispositivo Audio non sono più caratterizzate dalla struttura Unit, ma da un numero chiamato *chiave d'assegnazione*, rappresentato dal parametro ioa_AllocKey della struttura IOAudio (occupa una word). Ogni volta che il dispositivo riceve e può soddisfare una richiesta d'assegnazione dei canali e il parametro ioa_AllocKey della struttura di I/O risulta azzerato, il dispositivo stesso genera una nuova chiave d'assegnazione per quei canali, e la comunica al task. Successivamente, quando il task interagisce con uno di essi, deve indicare nelle richieste di I/O la relativa chiave d'assegnazione. Questa chiave è di esclusiva proprietà del task e a ogni canale non può corrispondere più di una chiave (mentre la stessa chiave può essere associata a diversi canali).

Ogni volta che chiede una nuova combinazione di canali, il task può

decidere se farsi assegnare una nuova chiave o utilizzare una chiave ottenuta in precedenza. In questo secondo caso, prima d'inviare il comando ADCMD_ALLOCATE deve memorizzare nel parametro ioa_AllocKey una delle chiavi d'assegnazione di cui è già in possesso. Mantenere un'unica chiave per tutti i canali che un task ottiene è utile per agire su di essi con un unico comando; infatti, se il task deve inoltrare lo stesso comando a diversi canali, e per ognuno di essi possiede una chiave diversa, si trova costretto a ripetere il comando tante volte quante sono le chiavi di cui dispone, come accade normalmente con gli altri dispositivi dell'Amiga.

Il parametro ioa_AllocKey non dev'essere confuso con l'array di combinazioni dei canali, e non è neanche il numero binario di 4 cifre che rappresenta i canali assegnati. È un numero generato internamente, costituito dal valore che il dispositivo ottiene incrementando di 1 l'ultima chiave d'assegnazione che ha dovuto creare; le chiavi che a mano a mano vengono generate costituiscono quindi una serie di numeri crescenti. Si tenga inoltre presente che le chiavi d'assegnazione non si ripetono mai (se non dopo 65.536 volte), e sono quindi uniche all'interno dell'intero sistema: due task che ottengono in tempi diversi l'accesso alla stessa combinazione di canali, ricevono chiavi d'assegnazione diverse.

Per comprendere meglio il significato e l'utilità pratica della chiave d'assegnazione, prendiamo come esempio una situazione ambigua in cui il dispositivo non saprebbe come cavarsela senza le chiavi d'assegnazione. Task1 ha indicato 50 come priorità, e ha ottenuto l'accesso ai canali 0 e 1 (la combinazione di canali numero 3); il dispositivo gli ha restituito la chiave d'assegnazione Chiave1, che mantiene anche internamente come chiave attuale. In seguito Task2 indica 51 come priorità e, tramite OpenDevice, richiede al dispositivo l'accesso alla stessa combinazione di canali. Il dispositivo rileva che per soddisfare questa richiesta l'unica via è sottrarre i canale 0 e 1 a Task1, e procede a confrontare le due priorità. La richiesta di Task2 è a maggiore priorità e quindi il dispositivo gli concede l'accesso creando una nuova chiave d'assegnazione, Chiave2, e memorizzandola anche internamente, al posto di Chiave1. Il dispositivo ora sa che la combinazione di canali costituita dai soli canali 0 e 1 è individuata da Chiave2.

A questo punto entrambi i task ritengono di avere libero accesso alla combinazione di canali 3, e quindi inviano richieste di I/O riferite sempre a quei canali: ognuno indica la propria chiave d'assegnazione nel parametro ioa_AllocKey, e imposta nel parametro io_Unit i bit relativi ai canali con cui desidera interagire (considerando che entrambi hanno aperto la combinazione di canali numero 3, possono indicare in io_Unit i valori 1, 2, e 3). Si suppone infatti che Task1 non abbia provveduto a inviare il comando ADCMD_LOCK, e quindi non si sia ancora reso conto che il suo accesso ai canali è stato sospeso (attenzione, non revocato). S'intuisce che per il dispositivo questa è una situazione ambigua, in quanto vede arrivare alla propria request port richieste che si riferiscono a canali della stessa combinazione, ma non ha strumenti che l'aiutino a distinguere quelle di Task1 (che per il momento non possono essere soddisfatte) da quelle di Task2 (che invece hanno il diritto di essere soddisfatte).

Ecco allora che la chiave d'assegnazione diventa lo strumento ideale per

distinguere tra le richieste da soddisfare e quelle che invece non possono essere soddisfatte. Task1 invia la propria richiesta indicando Chiave1; il dispositivo confronta la chiave indicata con quella che mantiene al suo interno (Chiave2) e rileva che sono diverse; quindi non accoglie la richiesta e la restituisce a Task1 indicando il codice d'errore ADIOERR_NOALLOCATION e azzerando il parametro io_Unit; solo adesso Task1 scopre che i canali 0 e 1 gli sono stati sottratti, e continua quindi a inviare la richiesta nella speranza che Task2 li liberi. Se invece la richiesta proviene da Task2, il dispositivo, accedendo al parametro ioa_AllocKey, rileva che la chiave d'assegnazione indicata è uguale a quella che mantiene internamente, e desume che la richiesta dev'essere soddisfatta.

La situazione illustrata si complica ulteriormente quando le combinazioni di canali richieste dai due task non sono identiche, e hanno per esempio solo il canale 0 in comune (combinazione di Task1 = 3; combinazione di Task2 = 9). In questo caso Task2 sottrae a Task1 solo il canale 0 quando invia il comando ADCMD_ALLOCATE con la priorità d'assegnazione 51, e le possibili situazioni diventano tre. Se Task1 invia per esempio il comando CMD_CLEAR al canale 1, il dispositivo accoglie ed esegue la richiesta. Se invece indica, tramite io_Unit, il solo canale 0, riceve dal dispositivo il codice d'errore ADIOERR_NOALLOCATION e il valore 0 nel parametro io_Unit. Infine, se indica nella richiesta entrambi i canali (io_Unit = 3), il dispositivo esegue il comando solo per il canale 1, e restituisce nel parametro io_Unit il valore 2; è questo valore a indicare a Task1 che CMD_CLEAR ha avuto successo solo con il canale 1.

Con questo esempio risulta evidente l'utilità della chiave d'assegnazione per discernere tra le richieste da soddisfare da quelle da ignorare. Tornando all'esempio, si noti che quando Task2 termina l'accesso alla combinazione di canali numero 3 inviando il comando ADCMD_FREE o eseguendo la funzione CloseDevice, se Task1 invia nuovamente la propria richiesta di I/O, il dispositivo può finalmente soddisfarla. Questo significa che Task1 non ha perso l'accesso a quei canali con l'arrivo di Task2, e non deve quindi riaprirli; è sufficiente che entri in un loop che continua a inviare la richiesta fino a quando non riceve il codice d'errore 0.

Con il dispositivo Audio i task devono sempre indicare nel parametro ioa_AllocKey delle loro strutture di I/O le chiavi d'assegnazione relative ai canali con i quali desiderano interagire, esattamente come dovrebbero fare (nel caso di altri dispositivi) con la struttura Unit. È proprio la possibilità d'inviare uno stesso comando a più unità contemporaneamente che costringe il dispositivo Audio a usare la chiave d'assegnazione al posto della struttura Unit.

Dall'esempio risulta anche evidente il ruolo svolto dal parametro io_Unit quando si inviano richieste di I/O a canali di cui si è già in possesso. I comandi previsti dal dispositivo Audio si aspettano infatti che oltre alla chiave d'assegnazione relativa alla combinazione richiesta, il task indichi nella struttura di I/O anche le unità su cui il comando deve agire (ovviamente, queste unità devono essere comprese nella combinazione per la quale il dispositivo ha restituito la chiave d'assegnazione). Possiamo quindi dire che la chiave d'assegnazione è una password per inviare comandi alle unità indicate dal parametro io_Unit, e concludiamo sottolineando il doppio ruolo giocato dal

parametro io_Unit nel dispositivo Audio: contiene sempre una combinazione di canali, che viene indicata dal task prima d'inviare un comando (escluso il comando d'assegnazione ADCMD_ALLOCATE), o restituita dal dispositivo per indicare i canali per i quali una particolare azione ha avuto successo. Come si può notare, due ruoli completamente diversi da quello standard descritto nei precedenti capitoli.

Come avvertire un task dell'imminente emissione di un suono

Può essere utile chiedere al dispositivo d'informare un task quando un canale audio inizia l'emissione di un suono. Una volta avvisato, il task può intraprendere attività (per esempio grafiche) che devono svolgersi in concomitanza con la produzione del suono. A questo scopo, il dispositivo Audio prevede nella struttura IOAudio la sotto-struttura ioa_WriteMsg, di tipo Message. Se si desidera che il task venga informato, tramite un messaggio, dell'imminente emissione di un suono (in seguito all'elaborazione di un comando CMD_WRITE da parte del dispositivo), occorre inizializzare il flag ADIOF_WRITEMESSAGE nel parametro io_Flags della struttura IOAudio che definisce il comando CMD_WRITE (si tenga presente che all'arrivo di questo messaggio nella reply port il task riceverà un segnale). Occorre infine ricordare sempre che il flag ADIOF_WRITEMESSAGE del parametro io_Flags viene utilizzato solo con il comando CMD_WRITE. I dettagli di questo sistema di comunicazione saranno analizzati più in dettaglio quando incontreremo il parametro ioa_WriteMsg nell'analisi della struttura IOAudio.

comandi del dispositivo Audio

Per programmare il dispositivo Audio si possono utilizzare gli otto comandi standard e sette comandi specifici del dispositivo. Elenchiamo ora alcuni accorgimenti fondamentali da tenere ben presenti quando s'inviano i comandi al dispositivo Audio.

- Per impartire qualsiasi comando è necessario inviare alle routine interne del dispositivo la relativa richiesta di I/O impiegando la funzione BeginIO e la struttura IOAudio. Quest'ultima è una struttura estesa non standard, nella quale il primo elemento è la struttura standard IORequest e i successivi costituiscono il messaggio.

- Per funzionare correttamente, tutti i comandi del dispositivo Audio esigono che il task indichi nella richiesta di I/O la chiave d'assegnazione relativa ai canali coinvolti. Per OpenDevice e ADCMD_ALLOCATE la chiave d'assegnazione può anche essere zero, mentre CloseDevice pretende, al pari degli altri comandi, un valore diverso da zero. Escludendo OpenDevice, CloseDevice, e ADCMD_ALLOCATE, tutti i comandi del dispositivo si aspettano che il task indichi nel parametro

io_Unit i canali sui quali devono agire; perché il comando possa agire contemporaneamente su più canali è necessario che questi possiedano la stessa chiave d'assegnazione. L'unica eccezione è costituita dal comando CMD_WRITE, che può agire su un solo canale per volta.

- La maggior parte dei comandi del dispositivo Audio sono sincroni: restituiscono il controllo al task solo a elaborazione completa, ovvero quando i task possono accedere alla struttura di I/O per controllare il risultato ottenuto. In particolare, vengono sempre trattati in modo sincrono tutti i comandi che il dispositivo può eseguire immediatamente, senza che intervengano attese. Se inviando questi comandi il task imposta il flag IOF_QUICK, il dispositivo non accoda la risposta alla reply port del task, e la comunicazione è molto più rapida; per questi comandi il QuickIO ha sempre successo, e si consiglia sempre di richiederlo, così da evitare l'accumularsi dei messaggi e l'obbligo di accedere alla propria reply port.

  Esistono altri quattro comandi che possono essere trattati dal dispositivo sia in modo sincrono sia in modo asincrono: CMD_WRITE, ADCMD_ALLOCATE, ADCMD_LOCK e ADCMD_WAITCYCLE. In particolare, questi comandi vengono trattati in modo asincrono se hanno successo, e in modo sincrono in caso contrario. Nel primo caso il flag IOF_QUICK viene azzerato e la risposta perviene alla reply port del task solo dopo un certo tempo. Nel secondo caso, invece, vale quanto detto pocanzi per i comandi sincroni. Si ricordi che per l'I/O asincrono è possibile utilizzare le funzioni CheckIO, WaitIO, GetMsg e Remove nella maniera consueta.

- Le richieste di I/O possono essere accodate, elaborate come QuickIO, oppure elaborate nel modo d'esecuzione immediato. La maggior parte dei comandi previsti dal dispositivo Audio vengono eseguiti subito dopo essere stati inviati dal task, salvo alcuni che possono richiedere un certo tempo per essere completati: ADCMD_FINISH, ADCMD_PERVOL (sempre sincroni) e la funzione AbortIO, che diventa asincrona se risulta inizializzato il flag ADIOF_SYNCCYLE (attesa dell'inizio del prossimo ciclo), il comando ADCMD_WAITCYCLE, che in maniera asincrona attende l'inizio di un nuovo ciclo nella riproduzione di una forma d'onda avviata con un comando CMD_WRITE, e infine il comando CMD_WRITE, che viene accodato in maniera asincrona se risulta in elaborazione un altro comando CMD_WRITE inviato dallo stesso task per gli stessi canali, o sospeso se è stato inviato il comando CMD_STOP.

- Alcuni comandi del dispositivo Audio possono essere utilizzati in codici di interrupt, ma soltanto con livelli di interrupt inferiori a 5.

- Tutti i comandi del dispositivo Audio restituiscono un codice d'errore nel parametro io_Error della struttura IORequest contenuta nella struttura IOAudio, e indicano nel parametro io_Unit i canali sui quali hanno agito.

La Figura 3.2 mostra in che modo vengono inviati i comandi alle routine interne del dispositivo Audio. Le linee con le frecce rappresentano i parametri che il task deve inizializzare e quelli restituiti dalle routine interne.

La Figura 3.2 descrive anche i parametri coinvolti nella preparazione e nell'elaborazione delle funzioni relative al dispositivo Audio. In particolare, le funzioni OpenDevice e CloseDevice influiscono sul parametro lib_OpenCnt della struttura Device. A questo proposito è bene aprire una parentesi sul particolare funzionamento del dispositivo Audio. Com'è noto dai capitoli precedenti, la funzione OpenDevice incrementa sempre il parametro lib_OpenCnt, e generalmente apre l'unità indicata nei suoi argomenti. Con la maggior parte dei dispositivi, OpenDevice costituisce l'unico strumento disponibile ai task per aprire le unità. A sua volta, la funzione CloseDevice decrementa sempre il parametro lib_OpenCnt, e generalmente chiude l'unità indicata nei suoi parametri. Anche in questo caso, con la maggior parte dei dispositivi CloseDevice costituisce l'unico strumento disponibile ai task per chiudere le unità che hanno aperto. Questi vincoli assicurano che i task chiamino sempre in egual misura le funzioni OpenDevice e CloseDevice, cioè che a ogni chiamata di OpenDevice corrisponda una e una sola chiamata a CloseDevice.

Con il dispositivo Audio la situazione si complica, in quanto i task possono aprire e chiudere le sue unità anche utilizzando i comandi ADCMD_ALLOCATE e ADCMD_FREE, e questi non modificano minimamente il parametro lib_OpenCnt della struttura Device. Può per esempio accadere che un task chiami due volte OpenDevice per aprire il dispositivo e richiedere l'accesso a due combinazioni di canali, e che successivamente liberi una delle due combinazioni con il comando ADCMD_FREE. In questo caso, quando poi libera anche l'altra combinazione chiudendo il dispositivo con la funzione CloseDevice, lib_OpenCnt risulterà decrementato solo una volta (anziché due) e questa è evidentemente una condizione anomala. Ovviamente la situazione può anche essere opposta: il task ha chiamato OpenDevice per aprire il dispositivo e accedere alla prima combinazione di canali, e ha inviato

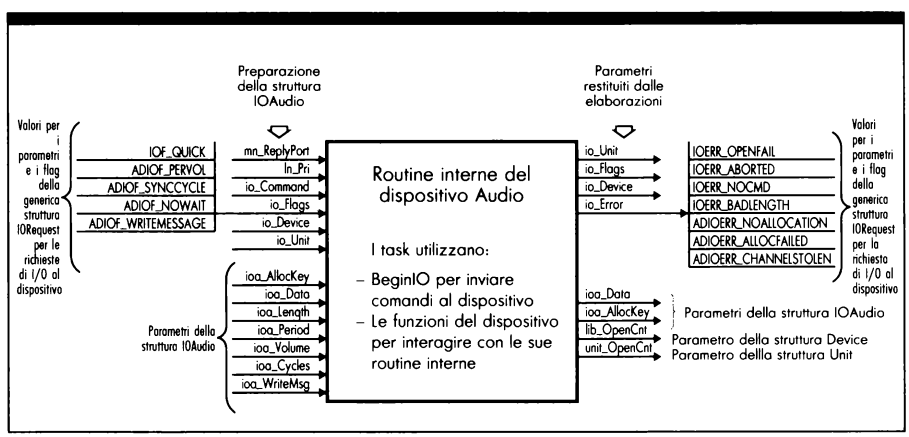


Figura 3.2: *Gestione delle funzioni e dei comandi previsti dal dispositivo Audio*

ADCMD_ALLOCATE per accedere alla seconda combinazione; successivamente ha chiuso le due combinazioni di canali chiamando due volte la funzione CloseDevice.

Per evitare queste anomalie, si consiglia di limitare l'impiego delle funzioni OpenDevice e CloseDevice allo stretto necessario. In pratica, se il task ha bisogno di una combinazione alla volta, conviene che si serva delle funzioni OpenDevice e CloseDevice non solo per aprire e chiudere il dispositivo, ma anche per ottenere e liberare i canali audio. Se invece il task ha l'esigenza di richiedere più combinazioni di canali nello stesso momento, è meglio che usi le funzioni OpenDevice e CloseDevice *solo* per aprire e chiudere il dispositivo, e usi invece i comandi ADCMD_ALLOCATE e ADCMD_FREE *solo* per assegnare e liberare combinazioni di canali.

Un altro problema si può presentare quando il task richiede l'accesso a diverse combinazioni di canali invocando sempre nuove chiavi d'assegnazione. Questo infatti può essere fonte di errori nella programmazione e in genere è del tutto inutile. La gestione dei canali diventa più facile se il task memorizza la chiave d'assegnazione che ottiene al momento della prima apertura del dispositivo, e la indica come parametro ogni volta che richiede nuovi accessi. In questo modo ogni task possiede un'unica chiave per tutto il tempo che tiene aperto il dispositivo.

Le strutture del dispositivo Audio

Il dispositivo Audio riconosce due strutture: IOAudio e AudChannel, illustrate nella Figura 3.3 (a pagina 95). La prima (anche se non è una struttura di I/O standard) è destinata a una gestione convenzionale delle interazioni con il dispositivo, mentre la seconda viene utilizzata per accedere direttamente ai registri hardware.

Con i comandi che funzionano in modo sincrono, il task può impiegare sempre la stessa struttura IOAudio, dal momento che ogni volta che la invia per far eseguire un comando, riottiene il controllo solo dopo che il dispositivo ha restituito la struttura in risposta: con i comandi sincroni ha sempre la certezza che accedendo alla struttura IOAudio dopo la chiamata alla funzione BeginIO non intacca un messaggio ancora in elaborazione. Con i comandi che invece possono essere trattati anche in maniera asincrona il task deve stare attento a non alterare il messaggio mentre il dispositivo lo sta elaborando: le conseguenze sono difficili da prevedere. Qualora si debbano inviare diverse richieste di I/O asincrone, conviene dunque che per ognuna il task impieghi una diversa struttura IOAudio, evitando così di compromettere messaggi in transito.

L'unica eccezione è costituita dal comando CMD_WRITE, che pur essendo asincrono (salvo errori) permette al task di alterare la struttura IOAudio subito dopo la chiamata alla funzione BeginIO. Questa caratteristica consente al task d'inviare subito un nuovo comando CMD_WRITE senza attendere la risposta al primo. Il secondo comando viene semplicemente accodato alla request port del dispositivo, e viene elaborato quando il primo si conclude.

In questo particolare caso, la possibilità di riutilizzare subito la struttura IOAudio nasce dal fatto che quando il dispositivo Audio la riceve (ovvero quando giunge alla sommità della coda) tutti i parametri che definiscono il suono da riprodurre vengono copiati nei registri hardware del canale richiesto, in modo che la riproduzione abbia inizio. Se dopo la chiamata a BeginIO si azzerano tutti i parametri della struttura che definiva il comando CMD_WRITE, i registri hardware rimangono inalterati, e quindi la riproduzione del suono non viene minimamente compromessa. Gli unici due parametri della struttura IOAudio che non vengono copiati dal dispositivo, e che quindi occorre mantenere integri, sono mn_ReplyPort, che individua il mittente del messaggio (si ricordi che se viene azzerato il dispositivo non restituisce il messaggio), e ioa_WriteMsg, che illustreremo fra breve.

Si badi però che quando un comando CMD_WRITE viene accodato, i suoi parametri vengono copiati negli opportuni registri hardware solo quando il dispositivo inizia a elaborarlo: per tutto il tempo che il comando rimane accodato, il task non deve accedere in scrittura ai parametri della relativa struttura IOAudio.

Comunque, anche se per il comando CMD_WRITE il dispositivo Audio permette una certa certa elasticità di gestione, è sempre meglio non alterare una struttura di I/O fino a quando non è stata restituita; attenersi alla gestione standard dei dispositivi è sempre più sicuro.

L'altra struttura, AudChannel, serve per accedere direttamente ai registri hardware che comandano i quattro canali audio dell'Amiga. Come vedremo, è organizzata per ricoprire esattamente i registri di un singolo canale, ed è definita nella struttura Custom, a sua volta definita nel file INCLUDE hardware/custom.h.

La struttura IOAudio

La struttura IOAudio è definita come segue:

```
struct IOAudio {
        struct IORequest ioa_Request;
        WORD ioa_AllocKey;
        UBYTE *ioa_Data;
        ULONG ioa_Length;
        UWORD ioa_Period;
        UWORD ioa_Volume;
        UWORD ioa_Cycles;
        struct Message ioa_WriteMsg;
};
```

I parametri della struttura IOAudio hanno il seguente significato:

- ioa_Request. È il nome della sotto-struttura di tipo IORequest i cui 6 parametri forniscono la rappresentazione standard del comando da

inviare al dispositivo. La struttura IORequest contiene una sotto-struttura di tipo Message, la quale contiene a sua volta il puntatore mn_ReplyPort, che deve puntare alla struttura MsgPort della reply port del task. Si ricordi che a questa message port il dispositivo invia le richieste di I/O in risposta, e che il task può possedere diverse reply port.

- ioa_AllocKey. Contiene la chiave d'assegnazione. Occupa una word e viene inizializzato dal dispositivo con una nuova chiave se durante l'elaborazione del comando ADCMD_ALLOCATE, o l'esecuzione della funzione OpenDevice, risulta azzerato. La chiave d'assegnazione dovrà poi essere copiata in tutte le strutture di I/O inviate per interagire con i canali relativi alla combinazione richiesta. Se si desidera inviare un comando che agisca su più canali contemporaneamente, a questi canali deve corrispondere la stessa chiave d'assegnazione; se i canali vengono aperti simultaneamente il task ottiene per forza un'unica chiave d'assegnazione, ma se sono stati aperti in tempi diversi (azzerando ogni volta il parametro ioa_AllocKey), il task non dispone di un'unica chiave d'assegnazione e quindi non può agire contemporaneamente su tutti i canali indicati; deve invece inviare il comando tante volte quante sono le chiavi d'assegnazione. Si ricordi che per ottenere l'accesso a nuovi canali senza che il dispositivo generi una nuova chiave d'assegnazione. occorre che nel parametro ioa_AllocKey della richiesta d'assegnazione sia indicata la chiave d'assegnazione precedente.

- ioa_Data. È un puntatore nel quale il task deve memorizzare l'indirizzo dell'array di dati (byte) che caratterizzano la richiesta di I/O. Svolge compiti diversi a seconda della funzione o del comando che viene inoltrato. Per esempio, quando si richiede l'accesso indicando diverse combinazioni di canali alternative, ioa_Data deve individuare in fast o chip RAM l'array di combinazioni di canali, nel quale ogni byte indica una delle combinazioni richieste. Quando invece si invia il comando

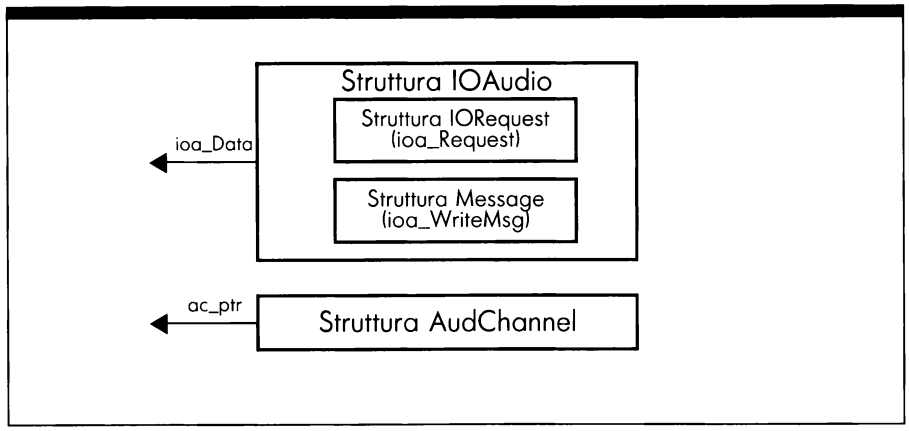


Figura 3.3: *Strutture utilizzate dal dispositivo Audio*

CMD_WRITE, ioa_Data deve individuare in memoria l'array che definisce la forma d'onda da riprodurre. Si ricordi che questo array deve risiedere nella chip RAM.

- ioa_Length. Contiene il numero di byte dell'array puntato dal parametro ioa_Data, cioè la quantità di dati che caratterizzano la richiesta di I/O. Questo parametro può avere diversi significati a seconda della funzione o del comando. Per esempio, quando si richiede l'accesso indicando diverse combinazioni di canali alternative, ioa_Length deve indicare la dimensione in byte dell'array di combinazioni dei canali individuato dal puntatore ioa_Data. Quando invece si invia il comando CMD_WRITE, ioa_Length deve indicare il numero di byte contenuti nell'array della forma d'onda (si ricordi che dev'essere un numero pari compreso tra 2 e 131.072).

- ioa_Period. Contiene il periodo di campionamento del suono da riprodurre, e il task lo inizializza quando invia il comando CMD_WRITE con il flag ADIOF_PERVOL impostato, o il comando ADCMD_PERVOL. Il periodo di campionamento è misurato in tick di sistema (un tick di sistema equivale a due cicli di clock, cioè a 279,365 ns nei sistemi americani e 281,932 ns nei sistemi europei), e sebbene teoricamente possa variare tra 1 e 65.535, è consigliabile per motivi di temporizzazioni del DMA non scendere a meno di 124. Il filtro anti-aliasing (adibito a eliminare le armoniche non desiderate che si formano quando la frequenza di campionamento ha uno scarto inferiore a 7 KHz rispetto alla frequenza audio) lavora con valori del periodo di campionamento inferiori a 300 o 500, a seconda della forma d'onda.

- ioa_Volume. Indica il volume del suono e dev'essere impostato dal task al momento dell'invio della richiesta di I/O; può variare tra 0 e 64 e viene utilizzato soltanto dai comandi CMD_WRITE (se è impostato il flag ADIOF_PERVOL) e ADCMD_PERVOL.

- ioa_Cycles. Indica quante volte dev'essere riprodotta la forma d'onda, e viene impostato dal task prima d'inviare il comando CMD_WRITE. Il periodo della ripetizione è ovviamente determinato da ioa_Period. Se il parametro ioa_Cycles è impostato a 0 la forma d'onda viene ripetuta all'infinito.

- ioa_WriteMsg. Questo è l'ultimo elemento della struttura IOAudio: contiene il nome di una sotto-struttura di tipo Message, e costituisce quindi un messaggio. Questa sotto-struttura rappresenta un'ulteriore possibilità di comunicazione con il dispositivo, ed è utilizzata dal comando CMD_WRITE. Se il task imposta il flag ADIOF_WRITEMESSAGE del parametro io_Flags quando invia il comando CMD_WRITE, le routine interne del dispositivo Audio restituiscono il messaggio ioa_WriteMsg al mittente in esso indicato nel momento esatto in cui inizia la riproduzione della forma d'onda. Non si

confonda questa struttura Message con l'altra struttura Message contenuta nella sotto-struttura IORequest e che costituisce l'intestazione del messaggio inviato al dispositivo. Anche la sotto-struttura IORequest viene inviata in risposta, ma nel momento in cui la riproduzione del suono termina.

La richiesta d'invio del messaggio ioa_WriteMsg può servire quando il task accoda diversi comandi CMD_WRITE nella request port del dispositivo, tutti indirizzati alla stessa combinazione di canali, e desidera sincronizzare un altro task (chiamiamolo TaskGrafico) con l'inizio di ogni suono perché per esempio muova i tasti di un pianoforte sullo schermo. Per farlo, in ognuna delle richieste di I/O memorizza nel parametro mn_ReplyPort del messaggio ioa_WriteMsg l'indirizzo della reply port di TaskGrafico, e imposta il flag ADIOF_WRITEMESSAGE; il dispositivo invierà a TaskGrafico un messaggio ioa_WriteMsg ogni volta che inizia l'esecuzione del relativo suono. Questo metodo di sincronizzazione è molto preciso, ed è indispensabile ogni volta che occorre iniziare un'operazione nell'esatto momento in cui inizia la generazione di un suono.

I valori che può assumere il parametro io_Flags della sotto-struttura IORequest contenuta nella struttura IOAudio, sono i seguenti:

- ADIOF_PERVOL. Si imposta questo flag se si desiderano nuovi valori per il periodo e per il volume. Lasciandolo a zero, il dispositivo utilizza nella generazione del suono gli ultimi valori che ha ricevuto, e quindi ignora i parametri ioa_Period e ioa_Volume della richiesta.

- ADIOF_SYNCCYCLE. Si imposta questo flag se si desidera che il dispositivo Audio porti a termine la riproduzione del suono che sta generando prima di eseguire l'azione indicata dalla richiesta di I/O (in genere si tratta dell'ordine d'immediato annullamento della riproduzione di una forma d'onda inviato tramite la funzione AbortIO, l'alterazione del periodo e del volume con il comando ADCMD_PERVOL, l'interruzione di un suono tramite il comando ADCMD_FLUSH). In questo modo, il task riottiene subito il controllo, ma il comando agisce solo alla fine del ciclo in esecuzione; nel caso della funzione AbortIO il controllo viene restituito solo quando l'annullamento della richiesta è stato effettuato. Il flag ADIOF_SYNCCYCLE può essere utilizzato per produrre effetti come il vibrato, il glissato, il tremolo, così come per cambiare il volume del suono. Generalmente, se si desidera unire senza discontinuità la fine di un suono con l'inizio di un altro, si fa in modo che l'ultimo dato campione della prima forma d'onda e il primo della nuova forma d'onda siano di ampiezze molto simili, di modo che attendendo la fine del ciclo e iniziando la riproduzione della seconda forma d'onda il passaggio da un suono al successivo sia il più possibile fluido. Si noti che il comando ADCMD_FINISH attende sempre la fine del ciclo in esecuzione prima di interrompere la riproduzione, anche se il flag ADIOF_SYNCCYCLE non è impostato.

- ADIOF_NOWAIT. Si imposta questo flag quando si invia il comando ADCMD_ALLOCATE per indicare al dispositivo che si desidera ottenere immediatamente l'accesso ai canali. Questo flag fa in modo che il task riottenga subito il controllo se nessuna delle combinazioni ha successo, per esempio quando per ogni combinazione uno o più canali sono detenuti da altri task con la medesima priorità (in questo caso, il parametro io_Error della struttura di I/O viene restituito con il codice d'errore IOERR_ALLOCFAILED). Se invece questo flag non viene impostato, il comando ADCMD_ALLOCATE non restituisce il controllo al task fino a quando una delle combinazioni non ha successo; in questo caso, per eliminare il comando si deve utilizzare la funzione AbortIO.

- ADIOF_WRITEMESSAGE. Si imposta questo flag se si desidera che il dispositivo restituisca il messaggio ioa_WriteMsg nel momento in cui inizia la generazione del suono richiesta con il comando CMD_WRITE.

La struttura AudChannel

La struttura IOAudio che abbiamo illustrato serve per accedere ai canali audio dell'Amiga senza comprometterne la gestione multitasking regolata dal dispositivo Audio. Il compito di questo dispositivo, infatti, è proprio quello d'interfacciare diversi task con gli stessi canali audio senza che avvengano conflitti, preoccupandosi cioè di regolare l'accesso ai registri hardware dei canali audio (questo è ovviamente il compito di tutti i dispositivi dell'Amiga, prescindendo dall'hardware con cui interagiscono). Talvolta possono però sorgere esigenze che richiedono un accesso diretto ai registri hardware dei canali audio, anche se si tratta di un'operazione che in un ambiente multitasking come l'Amiga non dovrebbe mai essere effettuata. Tuttavia, abbiamo visto che il dispositivo Audio permette a un task di proteggere i canali di cui è in possesso tramite il comando ADCMD_LOCK, il quale garantisce che nessun altro task acceda a quei canali finché non viene impartito il comando ADCMD_FREE. Il vantaggio di questo comando è che anziché bloccare la gestione multitasking dell'intero sistema, sospende solo la gestione multitasking di alcuni canali.

Per accedere ai canali audio direttamente, il task si può servire della struttura Custom, e più in particolare dell'array di quattro sotto-strutture AudChannel in essa contenute. Questa struttura è definita nel file INCLUDE hardware/custom.h, nel quale è definita anche una macro di nome custom che rappresenta una struttura Custom il cui primo elemento individua in memoria l'indirizzo 0xDFF000 (si tratta di una macro che viene utilizzata dal preprocessore). Questo è l'indirizzo di memoria a partire dal quale sono localizzati i registri hardware dei chip custom. Per agire direttamente su di essi, il task deve semplicemente impiegare questa macro per individuarli (dopo aver inviato il comando ADCMD_LOCK). Il programmatore può quindi redigere il sorgente in linguaggio C senza conoscere gli offset dei registri rispetto all'indirizzo 0xDFF000, e utilizzando per essi nomi simbolici.

Di tutti i registri hardware ai quali è possibile accedere in questo modo a

noi interessano solo quelli relativi ai canali audio. Osservando la definizione della struttura si nota che al suo interno è definito l'array aud[], nel quale ognuno dei quattro elementi è una struttura AudChannel e corrisponde in memoria ai registri hardware di un canale audio. Indicando l'elemento di questo array corrispondente a un particolare canale audio, il task accede ai relativi registri hardware. Prima di chiarire meglio le cose con un semplice esempio, analizziamo la struttura AudChannel.

```
struct AudChannel {
    UWORD *ac_ptr;
    UWORD ac_len;
    UWORD ac_per;
    UWORD ac_vol;
    UWORD ac_dat;
    UWORD ac_pad[2];
};
```

I parametri della struttura AudChannel hanno i seguenti significati:

- ac_ptr. Corrisponde al registro hardware che deve contenere l'indirizzo dell'array di dati campione che definiscono la forma d'onda per il particolare canale audio. Si ricordi che questo array deve risiedere nella chip RAM.

- ac_len. Corrisponde al registro hardware che deve contenere il numero di dati campione che costituiscono l'array individuato da ac_prt; questo numero dev'essere espresso in word (al contrario del parametro ioa_Length della struttura IOAudio, che contiene un numero espresso in byte).

- ac_per. Corrisponde al registro hardware che deve contenere il periodo di campionamento da impiegare nella riproduzione dei dati che definiscono la forma d'onda.

- ac_vol. Corrisponde al registro hardware che deve contenere il volume da impiegare nella riproduzione della forma d'onda.

- ac_dat. Corrisponde al registro hardware che durante la riproduzione del suono viene periodicamente aggiornato dal DMA con le coppie di dati campione da riprodurre. L'aggiornamente avviene a un ritmo doppio rispetto a quello indicato da ac_per, dal momento che il DMA copia dalla RAM due dati campione alla volta. Si ricordi che ogni dato campione può variare fra −128 e +127, estremi compresi.

- ac_pad[2]. Corrisponde a due registri hardware inutilizzati, il cui unico scopo è fare in modo che la nuova serie di registri per il canale audio successivo inizi a un indirizzo allineato con le long word.

Vediamo ora come si impiega la struttura AudChannel, ad esempio per agire sul registro del periodo di campionamento relativo al canale audio 0 e creare l'effetto vibrato. Nell'esempio si suppone che il task abbia già inviato un comando CMD_WRITE per riprodurre, con un periodo di campionamento pari a 504, una forma d'onda definita da 16 dati campione immagazzinati nella chip RAM, e che abbia già provveduto a inviare il comando ADCMD_LOCK per proteggere il canale. Il ciclo for() continua a modificare per un certo tempo il registro del periodo, incrementandolo linearmente per 50 volte, e decrementandolo linearmente per altre 50 volte (in questo modo non si fa altro che creare una modulazione di frequenza utilizzando una forma d'onda a triangolo, che corrisponde all'effetto vibrato).

Al termine del ciclo for() il vibrato termina, e si suppone che successivamente il task invierà il comando ADCMD_FREE per liberare il canale, in modo che il dispositivo possa tornare a gestirlo in modo multitasking.

```
#include <hardware/custom.h>
#include ...
struct IOAudio *iOAudio;

main()
{
     UWORD esterno;
     UWORD periodo;
     BYTE interno, incremento;
     ...
     periodo = iOAudio->ioa_Period;
     incremento = 1;
     for (esterno = Ø ; esterno < 5ØØØ ; esterno++);
     {
          for (interno = Ø ; interno < 1ØØ ; interno++);
          if (!(esterno % 5Ø)) incremento = -incremento;
          custom.aud[Ø].ac_per = (periodo += incremento);
     }
     ...
}
```

Si noti che l'esempio intende semplicemente mostrare un uso dell'accesso diretto ai registri hardware del controllo audio tramite le strutture Custom e AudChannel, e quindi non scende in altri dettagli che comunque non devono essere sottovalutati; per esempio, il ciclo for() interno di ritardo, in un ambiente multitasking come l'Amiga dev'essere assolutamente evitato, magari impiegando al suo posto il dispositivo Timer e l'attesa dei segnali.

Codici d'errore del dispositivo Audio

I messaggi d'errore restituiti dal dispositivo Audio sono i seguenti:

- ADIOERR_NOALLOCATION. Questo errore viene generato dal dispositivo durante l'impiego dei canali (e non quando vengono assegnati) se il dispositivo rileva che uno o più dei canali indicati dal parametro io_Unit non risultano accessibili al task, oppure se la chiave d'assegnazione non corrisponde a quella attuale. I canali indicati nella richiesta potrebbero risultare non accessibili perché il task non ha provveduto ad assegnarli o perché sono stati sottratti da una richiesta d'assegnazione con priorità più alta. Per esempio, se un task tenta d'inviare un comando a una combinazione di canali per i quali ha richiesto e ottenuto l'accesso (e la relativa chiave d'assegnazione), e uno o più di questi canali vengono sottratti da un altro task che ha indicato una priorità d'assegnazione maggiore, il dispositivo esegue il comando sui canali della combinazione che non risultano sottratti, restituisce l'errore ADIOERR_NOALLOCATION nel parametro io_Error, e azzera nel parametro io_Unit i bit corrispondenti ai canali della combinazione sui quali non ha potuto eseguire il comando. Quando il task accede alla risposta, riesce a scoprire l'esito del comando leggendo il contenuto dei parametri io_Unit e io_Error.

- ADIOERR_ALLOCFAILED. Il tentativo d'assegnazione di una combinazione di canali è fallito. Questo codice d'errore viene restituito dalla funzione OpenDevice (sia nel valore restituito, sia nel parametro io_Error della richiesta) quando tenta senza successo di assegnare canali. Anche il comando ADCMD_ALLOCATE restituisce questo codice in caso d'insuccesso, ma solo se il task ha impostato nella richiesta di I/O il flag ADIOF_NOWAIT. Si ricordi che una richiesta di assegnazione fallisce se per ognuna delle combinazioni indicate nell'array anche un solo canale risulta assegnato con più alta priorità a un altro task.

  Se invece durante una richiesta di assegnazione anche uno solo dei canali risulta protetto da un task con minore priorità d'assegnazione, non viene restituito il codice d'errore ADIOERR_ALLOCFAILED, dal momento che in questo caso il dispositivo si aspetta la liberazione, prima o poi, del canale protetto. In particolare, se la richiesta d'assegnazione viene inviata con la funzione OpenDevice, il controllo non viene restituito fino a quando il canale protetto non viene liberato, e quindi il task entra in attesa. Se invece la richiesta d'assegnazione viene effettuata tramite il comando ADCMD_ALLOCATE con il flag ADIOF_NOWAIT impostato, questo comando, rilevando che uno dei canali richiesti è protetto, restituisce subito il controllo al task.

- ADIOERR_CHANNELSTOLEN. Se un task ha protetto uno dei suoi canali tramite il comando ADCMD_LOCK, quando viene inoltrata al

dispositivo una richiesta d'assegnazione a più alta priorità per quel canale, il task che l'ha protetto riceve la risposta al comando ADCMD_LOCK e il codice d'errore ADIOERR_CHANNELSTOLEN nel parametro io_Error. In realtà non si tratta di un codice d'errore, ma di un avvertimento. Quando il task riceve questo codice sa che quel canale è richiesto con urgenza, e quindi deve al più presto liberarlo. Si noti che se il task non libera il canale questo resta protetto da qualsiasi azione intrapresa da altri task anche dopo l'invio del codice d'errore ADIOERR_CHANNELSTOLEN. Comunque, la richiesta d'assegnazione a più alta priorità non viene revocata, ma solo sospesa, come abbiamo spiegato illustrando il codice d'errore ADIOERR_ALLOCFAILED.

Sono previsti inoltre i quattro codici d'errore comuni a tutti i dispositivi dell'Amiga:

- IOERR_OPENFAIL. Indica che una chiamata alla funzione OpenDevice è fallita in quanto il sistema, per qualche ragione, non è riuscito ad aprire il dispositivo. Generalmente questa condizione d'errore si verifica (tanto per i dispositivi su disco quanto per quelli residenti in ROM) quando non è disponibile abbastanza memoria per aprire il dispositivo. Se si verifica, conviene liberare memoria ed effettuare nuovamente la chiamata a OpenDevice. Si ricordi che con il dispositivo Audio la funzione OpenDevice può non avere successo anche perché l'operazione di assegnazione è fallita per una delle ragioni che abbiamo già illustrato, cioè ragioni che riguardano la disponibilità dei canali. In questo caso, però, non viene restituito IOERR_OPENFAIL.

- IOERR_ABORTED. Il dispositivo restituisce questo errore solo quando il task chiama la funzione AbortIO per rimuovere dalla coda alla request port del dispositivo una particolare richiesta di I/O. Una volta che si sono ristabilite le condizioni necessarie, il task può azzerare il parametro io_Error e inviare nuovamente la richiesta di I/O. Si noti che se AbortIO viene mandata in esecuzione indicando una richiesta che si trova già in fase di elaborazione, l'elaborazione viene interrotta, ma non viene restituito il codice d'errore IOERR_ABORTED.

- IOERR_NOCMD. Il task ha inserito nel parametro io_Command un comando non riconosciuto dalle routine interne del dispositivo. Il task deve ridefinire il parametro e inviare nuovamente la richiesta.

La definizione precisa di questi errori comuni a tutti i dispositivi appare nel file INCLUDE denominato exec/errors.h. I nomi degli errori in esso riportati rappresentano valori distinti (compresi tra −1 e −4) che il sistema può assegnare al parametro io_Error di una struttura di I/O prima che venga restituita. Si noti che nell'elenco manca il codice d'errore IOERR_BADLENGTH.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOAudio)
A1

Scopo della funzione

Questa funzione chiude l'accesso da parte del task ai canali audio indicati dalla chiave d'assegnazione memorizzata nel parametro ioa_AllocKey della struttura IOAudio del task. Se non vi sono altri canali aperti dallo stesso task, CloseDevice chiude l'accesso da parte del task all'intero dispositivo. Questa funzione imposta sempre a −1 il puntatore io_Device della struttura di I/O ricevuta come argomento, e quindi se il task possiede altre chiavi d'assegnazione (cioè altri canali aperti), prima di riutilizzare la struttura IOAudio indicata nella chiamata a CloseDevice deve preoccuparsi di riaggiornare io_Device con l'indirizzo della struttura Device che aveva ottenuto aprendo il dispositivo. Comunque, per evitare problemi come questo, è consigliabile che i task allochino una struttura di I/O per ogni combinazione di canali per la quale richiedono una nuova chiave d'assegnazione.

Se il task ha accesso a più combinazioni di canali, e per ognuna ha una diversa chiave d'assegnazione, per chiudere completamente il dispositivo deve chiamare CloseDevice tante volte quante sono le chiavi d'assegnazione che possiede.

Argomenti della funzione

iOAudio — Dev'essere l'indirizzo di una struttura di tipo IOAudio opportunamente inizializzata.

Preparazione della struttura IOAudio

La struttura IOAudio può essere stata inizializzata da una precedente chiamata alla funzione OpenDevice, oppure semplicemente copiata da un'altra struttura di tipo IOAudio inizializzata sempre con la funzione OpenDevice. Oltre al consueto parametro io_Device, occorre che sia opportunamente inizializzato anche il parametro ioa_AllocKey con la chiave d'assegnazione relativa alla combinazione di canali che si desidera chiudere. Si noti che il parametro io_Unit non viene preso in considerazione, contrariamente a quanto accade con gli altri dispositivi; CloseDevice provvede semplicemente ad azzerarlo, qualunque sia il suo valore.

Discussione

CloseDevice chiude l'accesso da parte del task ai canali individuati dalla chiave d'assegnazione che il task deve aver memorizzato nel parametro ioa_AllocKey. Questa funzione decrementa sempre il parametro lib_OpenCnt della struttura Device. Come abbiamo già spiegato, nel caso del dispositivo Audio possono sorgere dei problemi se il task effettua un diverso numero di chiamate a OpenDevice e a CloseDevice. Può farlo perché, a differenza degli altri, il dispositivo Audio permette di aprire e chiudere le sue unità anche con appositi comandi. La possibile conseguenza è un valore del parametro lib_OpenCnt non corrispondente alla realtà. Anche se non si tratta di un problema grave (il dispositivo infatti è residente su ROM e non viene mai disallocato), è comunque opportuno programmare seguendo i consigli che abbiamo già illustrato.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("audio.device", ØL, iOAudio, ØL)
DØ                  AØ              DØ A1      D1
```

Scopo della funzione

Questa funzione apre l'accesso per il task al dispositivo Audio. Se la richiesta ha successo, OpenDevice imposta il parametro io_Device della

struttura IORequest in modo che punti a una struttura di tipo Device che il sistema utilizza per gestire il dispositivo.

Se il parametro ioa_Length della struttura IOAudio risulta diverso da 0, significa che il task ha memorizzato nel parametro ioa_Data l'indirizzo di un array di combinazioni di canali, e quindi che desidera l'assegnazione di alcuni canali (oltre a voler aprire il dispositivo). In questo caso, OpenDevice esegue automaticamente il comando ADCMD_ALLOCATE per assegnare al task una delle combinazioni di canali indicate nell'array, e segnala nel parametro io_Unit la combinazione che ha avuto successo. Si noti che OpenDevice, come del resto il comando ADCMD_ALLOCATE, non procede mai ad assegnazioni parziali: se in ogni combinazione indicata risultano uno o più canali inaccessibili, OpenDevice non assegna niente e restituisce il controllo al task indicando il codice d'errore ADIOERR_ALLOCFAILED sia nella variabile eguagliata alla funzione, sia nel parametro io_Error della struttura di I/O. Se invece uno dei canali richiesti risulta protetto da un task con più bassa priorità d'assegnazione, OpenDevice mette il task in attesa e restituisce il controllo solo quando quel canale viene liberato.

Se il parametro ioa_Length risulta diverso da zero, OpenDevice richiede per il task una reply port opportunamente inizializzata, eventualmente inserendo nel parametro mp_SigBit il numero di segnale relativo al bit che nel parametro tc_SigRecvd della struttura Task viene impostato a 1 ogni volta che alla reply port giunge un messaggio.

Quelli che seguono sono i dati restituiti dalla funzione OpenDevice.

- io_Device. Viene inizializzato con l'indirizzo della struttura di tipo Device che gestisce il dispositivo Audio. La struttura Device rimane sempre la stessa, per tutto il tempo che il dispositivo si trova in memoria, e contiene le informazioni necessarie per accedere a tutti i dati e alle routine contenute nella sua libreria.

- io_Unit. Questo parametro contiene una maschera nella quale i bit a 1 indicano i canali assegnati. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati. Esaminando io_Unit un task può rilevare quale combinazione di canali ha avuto successo nella procedura di assegnazione. Io_Unit risulta sempre 0 se il parametro ioa_Length è stato impostato a 0, cioè se non è stata richiesta l'apertura di nessuna unità.

- ioa_AllocKey. Il dispositivo restituisce in questo parametro una nuova chiave d'assegnazione qualora il task prima di chiamare OpenDevice l'abbia azzerato. La chiave d'assegnazione generata dal dispositivo è unica per tutto il tempo che il sistema rimane attivo, e dev'essere sempre indicata dal task quando invia un comando al dispositivo Audio. Quando il task desidera indicare particolari unità, oltre alla relativa chiave d'assegnazione deve impostare nel parametro io_Unit i bit corrispondenti ai canali sui quali il comando deve agire. Si noti che un task non può possedere uno stesso canale in due combinazioni diverse, cioè associato a due chiavi d'assegnazione, in quanto il dispositivo non consente la sovrapposizione delle combinazioni di canali.

- io_Error. Il codice restituito in questo parametro indica se l'apertura del dispositivo Audio, ed eventualmente l'assegnazione di alcune unità, ha avuto successo. Il valore 0 indica che l'operazione è avvenuta correttamente. Il codice d'errore IOERR_OPENFAIL indica che non è stato possibile aprire il dispositivo (per esempio nel sistema non c'era memoria libera sufficiente per allocare i buffer del dispositivo). Il codice d'errore ADIOERR_ALLOCFAILED indica invece che il dispositivo è stato aperto, ma non è stato possibile assegnare nessuna delle combinazioni di canali indicate dal task.

Argomenti della funzione

"audio.device" Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Audio.

ØL Dal momento che per indicare al dispositivo Audio le unità da aprire si utilizza l'array di combinazioni dei canali, questo parametro, a differenza di quanto accade negli altri dispositivi, non viene preso in considerazione.

iOAudio Il task deve indicare in questo argomento l'indirizzo alla struttura di tipo IOAudio che intende usare per aprire il dispositivo.

ØL Questo valore indica che l'argomento dei flag non viene considerato dal dispositivo Audio.

Preparazione della struttura IOAudio

Si devono inizializzare i seguenti parametri:

- ln_Pri. S'inizializza questo parametro solo se si desidera aprire il dispositivo Audio e contemporaneamente assegnarne le unità (il parametro ioa_Length è diverso da 0).

- mn_ReplyPort. Si deve inizializzare questo parametro in modo che punti a una struttura di tipo MsgPort che rappresenta la reply port del task al quale il dispositivo restituirà la risposta al termine dell'elaborazione. È necessario inizializzare questo parametro solo se si desidera aprire il dispositivo Audio e contemporaneamente assegnarne le unità (questo implica che il parametro ioa_Length è diverso da zero). Il task richiedente dovrà poi accertare, analizzando la risposta alla richiesta di I/O, quale combinazione di canali ha avuto successo.

- ioa_Data. Si deve inizializzare questo parametro con l'indirizzo dell'array di combinazioni di canali solo se il parametro ioa_Length è diverso da 0. In quest'array ogni byte deve rappresentare una combinazione di canali che soddisfa le esigenze del task. Anche se per definire una combinazione sono sufficienti 4 bit, si ricordi che ogni combinazione deve occupare un intero byte.

- ioa_Length. Si deve inizializzare questo parametro con il numero di combinazioni memorizzate nell'array di combinazioni di canali (un valore compreso tra 0 e 15). Se con la chiamata alla funzione non si desidera assegnare alcun canale, questo parametro dev'essere lasciato a zero.

- ioa_AllocKey. Si deve impostare a zero questo parametro se si desidera che il dispositivo generi una nuova chiave d'assegnazione. Altrimenti si deve inizializzarlo con una chiave assegnata in precedenza. In questo secondo caso, se il task ha anche richiesto l'assegnazione di alcuni canali e questa richiesta ha successo, a quei canali viene associata la chiave d'assegnazione indicata dal task.

Discussione

OpenDevice apre al task l'accesso al dispositivo ed eventualmente a una combinazione di canali indicata nell'array puntato da ioa_Data. Questa funzione incrementa sempre il parametro lib_OpenCnt della struttura Device. Come abbiamo già spiegato, nel caso del dispositivo Audio possono sorgere dei problemi se il task effettua un diverso numero di chiamate a OpenDevice e a CloseDevice. Può farlo perché a differenza degli altri, il dispositivo Audio permette di aprire e chiudere le sue unità anche con appositi comandi. La possibile conseguenza è un valore del parametro lib_OpenCnt non corrispondente alla realtà. Anche se non si tratta di un problema grave (il dispositivo infatti è residente su ROM e non viene mai disallocato), è comunque opportuno programmare seguendo i consigli che abbiamo già illustrato.

Quando OpenDevice apre con successo il dispositivo Audio, questo genera una nuova chiave d'assegnazione se nella relativa struttura di I/O il parametro ioa_AllocKey risulta azzerato. Si noti che genera una nuova chiave anche quando il parametro ioa_Length è impostato a 0, cioè quando non è richiesta l'assegnazione di alcun canale. In quest'ultimo caso, la chiave d'assegnazione ottenuta, apparentemente inutile, può comunque essere indicata come parametro quando si manda in esecuzione il comando ADCMD_ALLOCATE, in modo che il dispositivo non generi una nuova chiave.

Se invece chiamando OpenDevice il task ha indicato nel parametro ioa_AllocKey un numero diverso da zero, il dispositivo lo impiega come chiave d'assegnazione per la combinazione di canali assegnata. In questo modo, se il task possiede già una chiave d'assegnazione ottenuta in precedenza per una certa combinazione, può fare in modo che alla nuova combinazione venga stessa chiave,

associata quella stessa chiave, così che il task possa riferirsi contemporaneamente alle due combinazioni.

COMANDI STANDARD DEL DISPOSITIVO

CMD_CLEAR

Scopo del comando

CMD_CLEAR è un comando in grado di agire su più canali audio contemporaneamente: quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit li indica come canali accessibili, CMD_CLEAR restituisce semplicemente il controllo senza indicare alcun codice d'errore.

Il comando CMD_CLEAR viene sempre trattato in modo sincrono, e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Contiene una maschera nella quale i bit restituiti a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit indica canali accessibili, il dispositivo restituisce il codice d'errore 0. Se anche solo una di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Il task, leggendo il contenuto del parametro io_Unit, può rilevare su quali canali l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali risultati disponibili (per i quali i corrispondenti bit nel parametro io_Unit non sono stati azzerati) il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire con il comando; deve impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a 0; impostare infine io_Command con il comando CMD_CLEAR.

Discussione

Vi sono due comandi standard dei dispositivi che influenzano direttamente i buffer interni delle unità: CMD_UPDATE e CMD_CLEAR. CMD_CLEAR di solito riporta le routine interne del dispositivo a uno stato di default senza effettuare il reset dell'intero dispositivo, e azzera i buffer interni relativi all'unità indicata. Nel caso del dispositivo Audio, questi due comandi non fanno nulla, dal momento che non sono presenti buffer interni di transito dei dati. Compaiono nella libreria del dispositivo solo per ragioni di compatibilità con la gestione standard.

CMD_FLUSH

Scopo del comando

CMD_FLUSH è in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit indica canali accessibili, CMD_FLUSH interrompe nelle unità indicate le riproduzioni di forme d'onda alla fine del ciclo in atto, anche se il flag ADIOF_SYNCCYCLE non è stato impostato. Successivamente rimuove dalla coda alla request port del dispositivo le eventuali richieste di comandi CMD_WRITE e ADCMD_WAITCYCLE inviate alle unità indicate.

Il comando CMD_FLUSH viene sempre trattato in modo sincrono, e la struttura di I/O ritorna nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e

riottiene il controllo. Non è opportuno utilizzare il comando CMD_FLUSH in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Questo parametro contiene una maschera nella quale i bit restituiti a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit indica canali accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Il task, leggendo il contenuto del parametro io_Unit, può rilevare su quali canali l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali risultati disponibili (per i quali i corrispondenti bit nel parametro io_Unit non sono azzerati) il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire con il comando; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a 0; impostare infine io_Command con il comando CMD_FLUSH.

Discussione

Vi sono tre comandi del dispositivo Audio che influenzano direttamente la riproduzione dei suoni nei canali: CMD_FLUSH, CMD_RESET e ADCMD_FINISH. CMD_FLUSH interrompe la riproduzione di suoni nelle unità indicate e rimuove dalla coda alla request port tutte le richieste di I/O in essa accodate (a differenza della funzione AbortIO che è rimuove una particolare richiesta di I/O e lascia inalterate tutte le altre). A questo proposito, si noti che il dispositivo Audio accoda le richieste di I/O relative ai comandi CMD_WRITE in un modo abbastanza particolare.

Contrariamente a quanto si potrebbe pensare, se il task invia alla stessa

unità due comandi CMD_WRITE, il secondo non viene praticamente mai accodato, perché il dispositivo – non appena finisce di copiare i parametri del primo nei registri del canale audio (inizio del suono) – si appropria subito dell'indirizzo della seconda struttura di I/O, quella che descrive i parametri relativi al secondo suono. In questo modo, il secondo comando CMD_WRITE non viene accodato in attesa che si completi o venga soppressa la riproduzione del primo suono. Se il task invia invece più di due comandi CMD_WRITE, il terzo e i successivi vengono accodati (a meno che la generazione del primo suono non si sia già conclusa). Per esempio, supponiamo che il task invii quattro comandi CMD_WRITE alla stessa unità, e la generazione del primo suono non sia ancora finita: se si analizza la coda alla request port si riscontra che la seconda richiesta non è presente, mentre lo sono la terza e la quarta. Inviando il comando CMD_FLUSH all'unità, la riproduzione in corso viene interrotta alla fine del ciclo e la relativa richiesta di I/O viene restituita al task; la terza e la quarta richiesta di I/O vengono rimosse dalla coda e restituite al task con il codice d'errore IOERR_ABORTED nel parametro io_Error, ma il secondo comando CMD_WRITE, che non si trovava nella coda, viene elaborato.

Questo modo di operare permette al dispositivo di risparmiare tempo quando al termine di un suono deve impostare i registri hardware con i valori indicati dal secondo comando CMD_WRITE, in modo che i due suoni non risultino separati da un attimo, se pur minimo, di silenzio. Come vedremo illustrando il comando CMD_WRITE, l'accodamento di questo tipo di richiesta serve appunto a rendere il più rapido possibile il passaggio da un suono al successivo.

Dato che il comando CMD_FLUSH ha risultati distruttivi, l'utente dovrebbe utilizzarlo soltanto se desidera riportare il dispositivo allo stato iniziale, rimuovendo tutte le richieste di I/O accodate e attive.

CMD_READ

Scopo del comando

CMD_READ è in grado di agire su un solo canale audio per volta, quello indicato dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit indica un canale accessibile, CMD_READ restituisce al task l'indirizzo della struttura IOAudio relativa al comando CMD_WRITE che ha avviato la riproduzione del suono in corso. Se non c'è alcuna riproduzione in corso nell'unità indicata, CMD_READ restituisce il valore 0.

Il comando CMD_READ viene sempre trattato in modo sincrono, e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e

riottiene il controllo. I risultati prodotti da questo comando sono:

- io_Unit. Contiene una maschera nella quale il bit restituito a 1 indica il canale per il quale l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit indica canali accessibili, il dispositivo restituisce il codice d'errore 0. Se anche solo una di queste due condizioni viene meno il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION e azzera il bit corrispondente al canale nel parametro io_Unit.

- ioa_Data. Il dispositivo memorizza in questo parametro l'indirizzo della struttura IOAudio del comando CMD_WRITE che ha avviato la riproduzione del suono in corso. Se risulta pari a 0 significa che nell'unità indicata non è in corso la riproduzione di alcuna forma d'onda.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa al canale audio sul quale intende agire con il comando; si deve impostare a 1 il bit del parametro io_Unit corrispondente al canale sul quale s'intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a zero; impostare infine io_Command con il comando CMD_READ.

Discussione

Vi sono sette comandi del dispositivo Audio che hanno a che fare con il comando CMD_WRITE: ADCMD_PERVOL, ADCMD_WAITCYCLE, CMD_STOP, CMD_START, ADCMD_FINISH, CMD_FLUSH e CMD_READ. Questi comandi svolgono un'azione che modifica il comportamento delle unità a cui si riferiscono, ad eccezione di CMD_READ, che si limita a restituire un puntatore a una struttura IOAudio che rappresenta il comando CMD_WRITE in esecuzione nell'unità prescelta.

Quando un task vuole sapere quale comando CMD_WRITE, tra quelli che ha inviato, è in elaborazione, può utilizzare il comando CMD_READ per saperlo. Si noti che CMD_READ può essere impiegato soltanto con i canali aperti e

disponibili; non permette infatti di risalire alla struttura di I/O che un altro task ha inviato per sottrarre il canale e far riprodurre un nuovo suono. CMD_READ restituisce sempre indirizzi relativi alle strutture di I/O inviate dal task che l'ha inoltrato.

CMD_RESET

Scopo del comando

CMD_RESET è un comando in grado di agire contemporaneamente su tutti i canali audio indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit indica canali accessibili, CMD_RESET azzera i registri hardware del dispositivo Audio relativi a quei canali, interrompendo così l'eventuale riproduzione di suoni. Inoltre azzera i bit che unendo fra loro due canali realizzano la modulazione in frequenza o in ampiezza, ripristina il vettore di interrupt audio, esegue il comando CMD_FLUSH per eliminare tutte le richieste di I/O che si trovano accodate alla request port del dispositivo, e riattiva i canali nei quali la riproduzione delle relative forme d'onda era stata sospesa tramite il comando CMD_STOP. Non libera invece i canali protetti con il comando ADCMD_LOCK, per i quali occorre inviare il comando ADCMD_FREE.

Il comando CMD_RESET viene sempre trattato in modo sincrono, e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo. Non si deve utilizzare il comando CMD_RESET in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Contiene una maschera nella quale i bit restituiti a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit indica canali accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION.

  Leggendo il contenuto del parametro io_Unit, il task può rilevare quali sono i canali per cui l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore

ADIOERR_NOALLOCATION, sui canali che risultavano disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ha ottenuto chiamando OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a zero; impostare infine io_Command con il comando CMD_RESET.

Discussione

Ci sono due comandi del dispositivo Audio che influenzano direttamente la coda alla request port del dispositivo per un insieme di canali: CMD_RESET e CMD_FLUSH. CMD_FLUSH elimina tutte le richieste di I/O accodate per i canali indicati e interrompe la riproduzione di una forma d'onda al termine del ciclo in atto, mentre CMD_RESET, oltre a chiamare CMD_FLUSH per rimuovere le richieste in attesa, interrompe istantaneamente l'eventuale generazione di un suono, ripristinando nei canali le condizioni di default; infine chiama CMD_START se in quei canali le riproduzioni dei suoni erano state sospese dal comando CMD_STOP. Le richieste di I/O che il comando CMD_RESET provvede a rimuovere dalla coda vengono restituite al task con il codice d'errore IOERR_ABORTED nel parametro io_Error.

CMD_START

Scopo del comando

CMD_START è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, il parametro io_Unit indica canali accessibili e questi canali risultano bloccati con CMD_STOP, CMD_START rimette in azione tutti i comandi CMD_WRITE che erano in attesa.

CMD_START agisce in modo che i canali vengano attivati tutti nello stesso istante, così da minimizzare la distorsione nel caso in cui vari canali stiano riproducendo la stessa forma d'onda e le loro uscite siano miscelate. Dal momento che CMD_WRITE avvia la riproduzione di un suono su un solo canale, generalmente si usa il comando CMD_STOP e successivamente il comando CMD_START per sincronizzare i diversi suoni.

Il comando CMD_RESET viene sempre trattato in modo sincrono e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito da un segnale e riottiene il controllo. Non si deve utilizzare il comando CMD_START in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Questo parametro contiene una maschera nella quale i bit restituiti a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit li indica come accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Leggendo il contenuto del parametro io_Unit, il task può rilevare quali sono i canali per cui l'operazione non ha avuto successo. Si noti che nonostante venga restituito dal dispositivo il codice d'errore ADIOERR_NOALLOCATION. sui canali che risultavano disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a 0; impostare infine io_Command con il comando CMD_START.

Discussione

Vi sono due comandi che si occupano di far partire e sospendere la riproduzione di forme d'onda nelle unità del dispositivo Audio: CMD_START e CMD_STOP. CMD_STOP sospende le riproduzioni per tutte le unità indicate, mentre CMD_START le riattiva. Si noti che se in un canale è in corso la riproduzione di una forma d'onda e per quel canale il task invia il comando CMD_STOP, la riproduzione viene istantaneamente interrotta, anche se il ciclo in atto non è stato completato dal DMA. Se in seguito si riattiva il canale con il comando CMD_START, la riproduzione viene ripresa partendo con un nuovo ciclo, dal momento che il dispositivo può riattivarla soltanto con il primo dato campione della forma d'onda.

CMD_STOP

Scopo del comando

CMD_STOP è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit li indica come accessibili, CMD_STOP in quei canali sospende immediatamente le eventuali riproduzioni in corso. Una volta che il canale è stato bloccato, il dispositivo continua automaticamente ad accodare le richieste per quei canali fino a quando non viene eseguito un comando CMD_START per farlo ripartire, oppure CMD_RESET per azzerarlo.

Il comando CMD_STOP viene sempre trattato in modo sincrono, e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito da un segnale e riottiene il controllo. Non si deve utilizzare il comando CMD_STOP in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Questo parametro contiene una maschera nella quale i bit restituiti a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit indica canali accessibili, il dispositivo restituisce il codice d'errore 0. Se anche solo una di queste due condizioni viene meno, il dispositivo restituisce

il codice d'errore ADIOERR_NOALLOCATION. Leggendo il contenuto del parametro io_Unit, il task può rilevare su quali canali l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali che risultavano disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire con il comando; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a zero; impostare infine io_Command con il comando CMD_STOP.

Discussione

Vi sono due comandi che si occupano di far partire e sospendere la riproduzione di forme d'onda nelle unità del dispositivo Audio: CMD_START e CMD_STOP. CMD_STOP sospende le riproduzioni per tutte le unità indicate, mentre CMD_START le riattiva. Si noti che se in un canale è in corso la riproduzione di una forma d'onda e per quel canale il task invia il comando CMD_STOP, la riproduzione viene istantaneamente interrotta, anche se il ciclo in atto non è stato completato dal DMA. Se in seguito si riattiva il canale con il comando CMD_START, la riproduzione viene ripresa partendo con un nuovo ciclo, dal momento che il dispositivo può riattivarla soltanto con il primo dato campione della forma d'onda.

Si noti che CMD_STOP non tiene conto del flag ADIOF_SYNCCYCLE del parametro io_Flags appartenente alla struttura IOAudio. Quindi, se un task desidera bloccare la riproduzione del suono alla fine del ciclo per un insieme di canali del dispositivo Audio, deve prima utilizzare il comando ADCMD_WAITCYCLE, e poi inviare il comando CMD_STOP.

CMD_UPDATE

Scopo del comando

CMD_UPDATE è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit indica canali accessibili, CMD_UPDATE non fa nulla e restituisce il controllo senza indicare alcun codice d'errore.

Il comando CMD_UPDATE viene sempre trattato in modo sincrono, e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito da un segnale e riottiene il controllo. Non si deve utilizzare il comando CMD_UPDATE in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Contiene una maschera nella quale i bit restituiti a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit indica canali accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Leggendo il contenuto del parametro io_Unit, il task può rilevare quali sono i canali per cui l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali che risultavano disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device di gestione del dispositivo che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro

io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a 0; impostare infine io_Command con il comando CMD_UPDATE.

Discussione

Vi sono due comandi standard dei dispositivi che influenzano direttamente i buffer interni delle unità: CMD_UPDATE e CMD_CLEAR. CMD_CLEAR azzera i buffer interni del dispositivo, mentre CMD_UPDATE ne trasferisce il contenuto all'hardware. Nel caso attuale questi due comandi non eseguono nulla, dal momento che il dispositivo Audio non mantiene buffer interni di transito dei dati. I due comandi compaiono nella libreria del dispositivo solo per ragioni di compatibilità con la gestione standard dei dispositivi.

CMD_WRITE

Scopo del comando

CMD_WRITE è un comando che agisce su un singolo canale. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a un canale correttamente assegnato al task e il parametro io_Unit lo indica come accessibile, CMD_WRITE avvia la riproduzione della forma d'onda secondo i valori impostati dal task nei parametri della struttura di I/O. Se è in corso un'altra riproduzione avviata sempre dal task, oppure se il canale è stato bloccato tramite il comando CMD_STOP, le richieste di scrittura vengono accodate.

Se non si verificano errori nell'avvio della riproduzione, il comando CMD_WRITE viene sempre trattato in modo asincrono. Quindi il dispositivo azzera il flag IOF_QUICK nel parametro io_Error e restituisce la richiesta di I/O alla reply port del task quando la riproduzione è terminata. La riproduzione si conclude nei seguenti casi: il task ha indicato nella richiesta un numero finito di riproduzioni della forma d'onda; il canale viene sottratto da una richiesta d'assegnazione a più alta priorità; il task invia uno dei comandi che interrompono la riproduzione della forma d'onda, come ADCMD_FINISH. Dal momento che in questi casi nella richiesta di I/O relativa al comando CMD_WRITE non viene indicato alcun codice d'errore, il task, nel caso della sottrazione (l'unico nel quale l'interruzione della forma d'onda avviene per cause che non sono sotto il suo controllo, e che quindi non può prevedere) può richiedere di essere avvisato dell'evento inviando il comando ADCMD_LOCK.

Se invece la richiesta di I/O si trova ancora in coda alla request port del dispositivo, e viene rimossa da comandi come CMD_FLUSH e CMD_RESET, il dispositivo restituisce nella reply port del task la struttura di I/O della richiesta

indicando nel parametro io_Error il codice d'errore IOERR_ABORTED.

Può invece accadere che la chiave d'assegnazione (ioa_AllocKey) corrisponda a canali che il task non ha aperto, oppure indicati come inaccessibili (tutti o in parte) dal parametro io_Unit. In questo caso il comando viene trattato in modo sincrono, il dispositivo ne restituisce la struttura di I/O nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags risulta azzerato e il task, se è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo. Il codice d'errore restituito nel parametro io_Error è ADIOERR_NOALLOCATION. Non si deve utilizzare il comando CMD_WRITE in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Contiene una maschera di bit: se il bit corrispondente al canale risulta azzerato significa che il comando non ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a un canale correttamente assegnato e il parametro io_Unit lo indica come accessibile, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Se invece il comando è in esecuzione (la riproduzione del suono è già stata avviata), e viene bloccata con comandi come ADCMD_FINISH, nella struttura di I/O non viene restituito alcun codice d'errore. Infine, se il comando è ancora accodato e viene rimosso da comandi come CMD_FLUSH, CMD_RESET, o dalla funzione AbortIO, il dispositivo restituisce nella struttura di I/O il codice d'errore IOERR_ABORTED.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa al canale audio sul quale intende agire; impostare a 1 il bit del parametro io_Unit corrispondente al canale sul quale intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a zero; impostare infine io_Command con il comando CMD_WRITE.

Si devono inoltre inizializzare i seguenti parametri:

- io_Flags. Si deve impostare il flag ADIOF_PERVOL se si desidera definire un nuovo periodo di campionamento e un nuovo volume per la

forma d'onda. Impostare il flag ADIOF_WRITEMESSAGE se si desidera che il dispositivo restituisca al mittente il messaggio ioa_WriteMsg nel momento esatto in cui inizia la riproduzione della forma d'onda. In questo caso, il task deve memorizzare nel parametro mn_ReplyPort del messaggio ioa_WriteMsg l'indirizzo di una reply port, anche appartenente a un altro task, alla quale giungerà l'avviso che la riproduzione ha inizio. Questo task potrebbe per esempio effettuare animazioni sullo schermo in concomitanza con l'arrivo dei messaggi ioa_WriteMsg alla sua reply port. Si deve infine impostare il flag IOF_QUICK se si desidera che in caso d'errore il dispositivo non restituisca nessuna risposta alla reply port del task. Si noti che questo flag viene preso in considerazione solo se l'errore è stato generato da valori dei parametri ioa_AllocKey e io_Unit non appropriati (cioè se viene restituito il codice d'errore ADIOERR_NOALLOCATION).

- ioa_Data. Si deve inizializzare questo parametro con l'indirizzo dell'array che definisce i dati campione della forma d'onda da riprodurre. Ogni dato campione di questo array è costituito da un byte il cui valore può variare tra −128 e +127, e rappresenta l'ampiezza della forma d'onda in un particolare istante. L'array della forma d'onda dev'essere allocato nella chip RAM e il suo primo elemento dev'essere allineato con le word.

- ioa_Length. Si deve inizializzare questo parametro con il numero di byte contenuti nell'array della forma d'onda. Dev'essere un numero pari compreso tra 2 e 131.072. Quantità maggiori non producono errori, ma vengono sottoposte all'operazione logica AND per azzerare i bit eccedenti.

- ioa_Period. Si deve inizializzare questo parametro con il nuovo valore del periodo di campionamento da utilizzare nella riproduzione della forma d'onda. Il periodo di campionamento viene misurato in tick di sistema (un tick di sistema equivale a due cicli di clock, e quindi corrisponde a un tempo di 279,365 ns nei sistemi americani, e 281,932 ns nei sistemi europei) e rappresenta il tempo che deve trascorrere fra la riproduzione di un dato campione e il successivo. Il suo valore può variare tra 124 e 65.535. Il filtro per l'eliminazione delle armoniche non desiderate lavora con valori inferiori a 300 o a 500, a seconda del tipo di forma d'onda. Ha senso specificare questo parametro soltanto se il flag ADIOF_PERVOL è stato impostato, altrimenti la forma d'onda viene comunque riprodotta impiegando l'ultimo periodo di campionamento impostato per quel canale.

- ioa_Volume. Si deve inizializzare questo parametro con il nuovo valore del volume per il canale indicato. Il volume può variare tra 0 (attenuazione massima) e 64 (attenuazione minima). Ha senso specificare questo parametro soltanto se il flag ADIOF_PERVOL è stato impostato, altrimenti la forma d'onda viene comunque riprodotta

impiegando l'ultimo livello di volume impostato per quel canale.

- ioa_Cycles. Si deve inizializzare questo parametro con il numero di ripetizioni previste per la forma d'onda; il numero può essere compreso tra 0 e 65.535. Con il valore 0 la forma d'onda viene ripetuta infinite volte.

- ioa_WriteMsg. Se si desidera far pervenire a una reply port un messaggio nell'esatto istante in cui inizia la riproduzione della forma d'onda indicata da ioa_Data, cioè se si è impostato il flag ADIOF_WRITEMESSAGE del parametro io_Flags, occorre inizializzare anche alcuni parametri della sotto-struttura ioa_WriteMsg. Trattandosi di una struttura di tipo Message, occorre memorizzare nel parametro mn_ReplyPort l'indirizzo a cui il messaggio ioa_WriteMsg deve pervenire, cioè l'indirizzo di una reply port. Questa reply port può appartenere al task che invia il comando, oppure a qualsiasi altro task. Oltre a mn_ReplyPort, il task può anche decidere di allocare in memoria alcuni byte subito dopo la struttura IOAudio, nei quali inserire eventualmente un messaggio per l'utente. In questo modo, non appena inizia la riproduzione della forma d'onda, la reply port indicata riceve prontamente il messaggio ioa_WriteMsg.

Discussione

CMD_WRITE è l'unico comando del dispositivo Audio che permette di dare il via alla riproduzione dei suoni. Un task può inviare un flusso continuo di comandi CMD_WRITE a ciascuna unità che ha aperto. Ognuno di essi viene accodato nella coda alla request port dell'unità interessata. Non essendo previsto il QuickIO per questo comando, i suoni vengono sempre riprodotti nell'ordine in cui sono stati inviati al dispositivo.

Tuttavia, questo non vuol dire che una volta accodato un comando CMD_WRITE, le operazioni del sistema non possano essere alterate: per esempio, i comandi CMD_FLUSH, CMD_RESET, AbortIO e CMD_STOP influenzano anche le richieste di I/O accodate. Inoltre, i comandi CMD_STOP e CMD_START possono sospendere e far riprendere la riproduzione di una forma d'onda. Infine il comando ADCMD_PERVOL può cambiare il periodo e il volume mentre è in corso la riproduzione di un suono.

Per quanto riguarda l'accodamento dei comandi CMD_WRITE, durante la programmazione occorre tener conto che il dispositivo rimuove il comando CMD_WRITE dalla sommità della coda quando inizia la riproduzione della forma d'onda indicata dal precedente comando CMD_WRITE. Questo significa che se s'inviano per esempio cinque comandi CMD_WRITE e dopo aver inoltrato l'ultimo la riproduzione del primo suono non è ancora terminata, nella coda risultano soltanto tre richieste di I/O. dal momento che la seconda è stata subito rimossa dal dispositivo quando ha iniziato la riproduzione del primo suono. Di questo metodo di gestione occorre tener conto quando si desidera per

esempio eliminare tutte le richieste accodate a una particolare unità, in quanto i comandi CMD_FLUSH e CMD_RESET interrompono la riproduzione in corso, rimuovono tutte le richieste di I/O accodate, ma non eliminano la richiesta di I/O relativa al comando CMD_WRITE inviato subito dopo quello in elaborazione. Per rimuovere anche quella, l'unico modo è chiamare la funzione AbortIO subito dopo il comando CMD_FLUSH o CMD_RESET; l'argomento della funzione AbortIO dev'essere ovviamente l'indirizzo della struttura di I/O relativa al secondo comando CMD_WRITE.

Questa particolare gestione delle richieste di I/O è definita double-buffered, e serve per ridurre al minimo il tempo che trascorre fra la fine di un suono e l'inizio del successivo, evitando tempi morti e la generazione di rumori indesiderati. I task possono sfruttare questa prerogativa inviando comandi CMD_WRITE a una velocità superiore a quella con cui vengono elaborati.

Impiegando il comando CMD_WRITE occorre infine tenere conto di un bug non risolto nemmeno con la versione 1.3 del software sistema: quando il dispositivo si trova a eseguire un comando CMD_WRITE subito dopo aver finito la riproduzione di una precedente forma d'onda, se non è stato impostato il flag ADIOF_PERVOL i parametri ioa_Data e ioa_Length del nuovo comando potrebbero essere ignorati.

COMANDI SPECIFICI DEL DISPOSITIVO

ADCMD_ALLOCATE

Scopo del comando

ADCMD_ALLOCATE serve per assegnare al task una combinazione di canali tra quelle indicate nell'array di combinazioni. Si tratta di un metodo alternativo a OpenDevice per ottenere l'accesso alle unità del dispositivo Audio.

ADCMD_ALLOCATE viene trattato in modo sincrono se una combinazione ha successo e nessun canale è protetto, oppure se nessuna delle combinazioni ha successo e il flag ADIOF_NOWAIT risulta impostato: in questi due casi il dispositivo restituisce la struttura di I/O solo se il flag IOF_QUICK è a zero.

In ogni altro caso ADCMD_ALLOCATE viene trattato in modo asincrono, cioè il dispositivo azzera il flag IOF_QUICK e restituisce la struttura di I/O soltanto quando una delle combinazioni ha successo e nessuno dei canali che indica è protetto. I task non devono utilizzare il comando ADCMD_ALLOCATE

nei codici di interrupt a qualsiasi livello. I risultati prodotti da questo comando sono i seguenti:

- io_Unit. In questo parametro viene indicata la combinazione di canali che ha avuto successo. Si ricordi che il comando, al pari della funzione OpenDevice, non assegna mai una combinazione parziale.

- io_Flags. Il flag IOF_QUICK viene azzerato se il dispositivo Audio tratta il comando ADCMD_ALLOCATE come una richiesta di I/O asincrono.

- io_Error. Un valore di questo parametro pari a 0 indica che il comando è stato eseguito con successo. Il codice d'errore ADIOERR_ALLOCFAILED indica che la richiesta d'assegnazione dei canali è fallita. Questo codice viene restituito soltanto quando nessuna delle combinazioni è assegnabile e il task ha impostato il flag ADIOF_NOWAIT.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ln_Pri con la priorità che desidera associare ai canali (un valore compreso tra −128 e +127) e impostare io_Command con il comando ADCMD_ALLOCATE. Deve inizializzare inoltre i seguenti parametri specifici del dispositivo.

- io_Flags. Si deve inizializzare questo parametro a 0, oppure a IOF_QUICK per ottenere il QuickIO. Se il QuickIO fallisce (per esempio almeno uno dei canali richiesti è protetto) le routine del dispositivo Audio trattano il comando ADCMD_ALLOCATE in modo asincrono: il dispositivo azzera il flag IOF_QUICK e restituisce la risposta quando riesce ad assegnare una delle combinazioni di canali indicate dal task. Si deve inizializzare io_Flags a ADIOF_NOWAIT se si desidera che il comando ADCMD_ALLOCATE restituisca subito il controllo qualora al primo tentativo nessuna delle combinazioni risulti assegnabile (in questo caso viene restituito il codice d'errore ADIOERR_ALLOCFAILED e la richiesta viene trattata in modo sincrono). Si noti che la richiesta non fallisce, e quindi non viene restituita subito, qualora alcuni dei canali risultino protetti.

- ln_Pri. Si deve inizializzare questo parametro con la priorità che si vuole attribuire alla combinazione di canali che ha successo.

- ioa_Data. Si deve inizializzare questo parametro con l'indirizzo dell'array di combinazioni dei canali. In quest'array ogni byte

rappresenta una combinazione di canali che soddisfa le esigenze del task. Anche se per definire una combinazione sono sufficienti 4 bit, si ricordi che ogni combinazione deve occupare un intero byte.

- ioa_Length. Si deve inizializzare questo parametro con il numero di combinazioni memorizzate nell'array (un valore compreso tra 0 e 15).

- ioa_AllocKey. Si deve inizializzare a 0 questo parametro se si desidera che il dispositivo generi una nuova chiave d'assegnazione. In caso contrario si deve indicare una precedente chiave d'assegnazione.

Discussione

Esistono due modi per assegnare i canali audio. Il primo utilizza la funzione OpenDevice, che provvede a chiamare il comando ADCMD_ALLOCATE. Il secondo modo è chiamare il comando ADCMD_ALLOCATE direttamente.

ADCMD_ALLOCATE analizza ogni combinazione di canali specificata dal task nell'array, alla ricerca di quella che non richiede la sottrazione di alcun canale (cioè quella che contiene solo canali disponibili); se nessuna delle combinazioni indicate ha successo, ADCMD_ALLOCATE cerca quella che sottrae ad altri task canali con la più bassa priorità possibile. Perché anche la seconda fase non abbia successo basta che uno solo dei canali (in ciascuna combinazione) risulti posseduto da un altro task con priorità uguale o maggiore.

Se anche con la sottrazione dei canali nessuna delle combinazioni ha successo, si distinguono due casi a seconda che il task abbia o meno impostato il flag ADIOF_NOWAIT prima d'inviare il comando: se è impostato, il comando azzera il parametro io_Unit, memorizza il codice d'errore ADIOERR_ALLOCFAILED nel parametro io_Error e restituisce la struttura di I/O alla reply port del task; se invece ADIOF_NOWAIT è azzerato, ADCMD_ALLOCATE ritenta l'assegnazione quando vengono liberati canali (tramite ADCMD_FREE o CloseDevice) o quando viene eseguito un comando ADCMD_SETPREC che altera le priorità dei canali, e restituisce la struttura di I/O solo quando una delle combinazioni viene assegnata.

Se una delle combinazioni indicate dal task ha successo, ADCMD_ALLOCATE procede a controllare che nessuno dei canali coinvolti dalla combinazione sia stato protetto da un comando ADCMD_LOCK. Se anche un solo canale risulta protetto, ADCMD_ALLOCATE restituisce la struttura di I/O relativa al comando ADCMD_LOCK indicando nel suo parametro io_Error il codice d'errore ADIOERR_CHANNELSTOLEN e si mette in attesa che quel canale venga liberato, anche se il flag ADIOF_NOWAIT è stato impostato. Quando tutti i canali risultano disponibili, ADCMD_ALLOCATE compie le seguenti operazioni. 1) Esegue su quei canali il comando CMD_RESET. 2) Se il parametro ioa_AllocKey contiene il valore zero, genera una nuova chiave d'assegnazione; altrimenti adotta la chiave d'assegnazione indicata dal task. 3) Copia la chiave d'assegnazione in tutti i canali della combinazione. 4) Copia la

priorità in tutti i canali della combinazione. 5) Copia la combinazione che ha avuto successo nel parametro io_Unit e restituisce al task la struttura di I/O.

ADCMD_ALLOCATE può operare sia in modo sincrono sia in modo asincrono, a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando. Quando viene trattato in modo asincrono, il task può eliminarlo chiamando la funzione AbortIO.

Quando ADCMD_ALLOCATE sottrae un canale posseduto da un altro task con una più bassa priorità d'assegnazione, i comandi CMD_WRITE che il primo task ha accodato a quell'unità vengono rimossi dalla coda, e nelle loro strutture di I/O viene indicato il codice d'errore IOERR_ABORTED. Si noti che il comando CMD_WRITE corrispondente alla riproduzione in corso viene interrotto e che il comando CMD_WRITE successivo, quello mantenuto internamente dal dispositivo, non viene eseguito. In entrambi i casi non viene restituito nessun codice d'errore.

Se si decide di lavorare direttamente con i registri hardware del dispositivo Audio in linguaggio Assembly, i canali audio assegnati da ADCMD_ALLOCATE devono essere protetti (tramite il comando ADCMD_LOCK), oppure occorre inizializzarne la priorità con il massimo valore possibile (+127), in modo da evitare che vengano sottratti da altri task.

ADCMD_FINISH

Scopo del comando

ADCMD_FINISH è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, il parametro io_Unit li indica come accessibili e in quei canali sono in corso riproduzioni di forme d'onda, ADCMD_FINISH interrompe le riproduzioni immediatamente (o al termine del ciclo in corso, nel caso che il task abbia impostato il flag ADIOF_SYNCCYCLE del parametro io_Flags).

Il comando ADCMD_FINISH viene sempre trattato in modo sincrono e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo. Non si deve utilizzare ADCMD_FINISH in codici di interrupt che lavorano a livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Questo parametro contiene una maschera nella quale i bit a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati e il parametro io_Unit li indica come accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Leggendo il contenuto del parametro io_Unit, il task può rilevare quali sono i canali per cui l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali risultati disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti azzerare il parametro io_Flags; impostare il flag ADIOF_SYNCCYCLE se desidera che la riproduzione del suono in ogni canale venga interrotta al termine del ciclo in corso; impostare infine io_Command con il comando ADCMD_FINISH.

Discussione

ADCMD_FINISH permette a un task d'interrompere la generazione di uno o più suoni che aveva avviato tramite CMD_WRITE. Se nei canali specificati non risulta in corso alcuna riproduzione, ADCMD_FINISH non ha alcun effetto. Il task può decidere se interrompere le riproduzioni immediatamente oppure alla fine dei cicli in atto rispettivamente azzerando o impostando il flag ADIOF_SYNCCYCLE del parametro io_Flags.

Anche la funzione AbortIO può eliminare un comando CMD_WRITE in esecuzione, ma non permette d'interrompere la riproduzione immediatamente, cioè si sincronizza sempre con la fine del ciclo. Inoltre, AbortIO è in grado d'interrompere la riproduzione di una sola forma d'onda, mentre ADCMD_FINISH è in grado di agire su più canali simultaneamente.

ADCMD_FREE

Scopo del comando

ADCMD_FREE è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit li indica come accessibili, ADCMD_FREE li libera, in modo che altri task possano ottenere l'accesso. Le operazioni che compie su quei canali sono le seguenti. 1) Ripristina le condizioni di default utilizzando CMD_RESET. 2) Cambia la chiave d'assegnazione. 3) Libera i canali che risultano protetti e azzera i bit corrispondenti ai canali liberati nel parametro io_Unit della struttura IOAudio che rappresenta la precedente richiesta ADCMD_LOCK; se la struttura IOAudio della richiesta ADCMD_LOCK non possiede alcun bit dei canali impostato nel proprio parametro io_Unit, ADCMD_FREE la restituisce. 4) Verifica l'esistenza di richieste di assegnazione in attesa: i canali ora sono disponibili per essere riassegnati tramite il comando ADCMD_ALLOCATE o la funzione OpenDevice.

Il comando ADCMD_FREE viene sempre trattato in modo sincrono e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo. Non si deve utilizzare ADCMD_FREE in codici di interrupt a qualsiasi livello.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Contiene una maschera nella quale i bit a 0 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit li indica come accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Leggendo il contenuto del parametro io_Unit, il task può rilevare quali sono i canali per cui l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali risultati disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ha ottenuto chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a 0; impostare infine io_Command con il comando ADCMD_FREE.

Discussione

Il comando ADCMD_FREE permette a un task di liberare i canali audio che ha precedentemente assegnato con la funzione OpenDevice o con il comando ADCMD_ALLOCATE. Una volta che i canali indicati sono stati liberati dal task, questo o altri task possono procedere ad assegnarli nuovamente.

Si noti che ADCMD_FREE libera anche i canali che sono stati protetti dal comando ADCMD_LOCK. Al posto del comando ADCMD_FREE, il task può anche impiegare la funzione CloseDevice, solo che in questo caso non può scegliere quali canali della combinazione chiudere e quali lasciare aperti.

ADCMD_LOCK

Scopo del comando

ADCMD_LOCK è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit li indica come accessibili, ADCMD_LOCK attiva una protezione che garantisce al task l'assoluto dominio su quei canali. In questo caso il comando viene trattato in modo asincrono, viene azzerato il flag IOF_QUICK e la richiesta viene restituita alla reply port del task solo quando tutti i canali sono stati liberati o quando viene fatta una richiesta d'assegnazione a più alta priorità per uno dei canali protetti.

Se anche una sola delle due condizioni non si verifica, ADCMD_LOCK viene trattato in modo sincrono e viene restituito alla reply port del task soltanto se il flag IOF_QUICK risulta azzerato. In questo caso, se il task è entrato in attesa

tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo. Non si deve utilizzare il comando ADCMD_LOCK in codici di interrupt a qualsiasi livello.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Contiene una maschera nella quale i bit a 1 indicano i canali che sono ancora protetti. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati e il parametro io_Unit li indica come accessibili, il dispositivo può indicare due diversi codici d'errore al momento della restituzione: se la restituzione della richiesta avviene in quanto tutti i canali sono stati liberati, il parametro io_Error riporta il codice d'errore 0 e io_Unit il valore 0. Se invece la richiesta viene restituita perché un altro task ha chiesto l'accesso a un canale indicando una priorità maggiore, il dispositivo restituisce il codice d'errore ADIOERR_CHANNELSTOLEN.

  Se invece la chiave d'assegnazione non corrisponde a canali correttamente assegnati al task, oppure il parametro io_Unit indica canali inaccessibili, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ha ottenuto chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; impostare a 1 i relativi bit del parametro io_Unit; impostare il flag IOF_QUICK del parametro io_Flags se vuole che in caso d'insuccesso la richiesta di I/O non venga restituita alla reply port del task; impostare infine io_Command con il comando ADCMD_LOCK.

Discussione

ADCMD_LOCK permette a un task di evitare che altri task gli sottraggano senza autorizzazione i canali audio a lui assegnati. Per ottenere questa protezione si potrebbe anche inizializzare la priorità dei canali al massimo valore tramite ADCMD_ALLOCATE. OpenDevice o ADCMD_SETPREC, con lo svantaggio però che nessun altro tentativo di assegnazione di quei canali sortirebbe effetto. Se invece si impiega il comando ADCMD_LOCK, tutti i task

che tentano di assegnare i canali audio con priorità maggiori entrano in stato di attesa fino a quando i canali protetti non vengono liberati dal task che ha impartito il comando ADCMD_LOCK. In definitiva, questo comando non impedisce ad altri task di impossessarsi dei canali: ne rallenta solo le assegnazioni. Si noti che il task che ha richiesto l'accesso a canali protetti entra in attesa sia che abbia impiegato il comando ADCMD_ALLOCATE con il flag ADIOF_NOWAIT impostato, sia che abbia impiegato OpenDevice con il parametro ioa_Length diverso da zero.

I canali protetti possono essere liberati eseguendo il comando ADCMD_FREE (il quale azzera il bit relativo a ogni canale che libera nel parametro io_Unit), oppure chiudendo la combinazione di canali con il comando CloseDevice. ADCMD_LOCK non viene restituito alla reply port del task fino a quando i canali protetti non vengono liberati, a meno che non si verifichi una richiesta dei canali protetti che abbia priorità maggiore, nel qual caso avvisa il task che l'ha chiamato restituendogli la struttura di I/O con il codice d'errore ADIOERR_CHANNELSTOLEN. Il task che ha protetto i canali deve allora provvedere a liberare al più presto il canale richiesto.

ADCMD_PERVOL

Scopo del comando

ADCMD_PERVOL è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, il parametro io_Unit li indica come accessibili e in quei canali risultano in corso riproduzioni di forme d'onda, ADCMD_PERVOL modifica il periodo e il volume per ogni canale indicato. La modifica può avvenire immediatamente, oppure alla fine del ciclo se il task ha impostato il flag ADIOF_SYNCCYCLE.

Il comando ADCMD_PERVOL viene sempre trattato in modo sincrono, e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo. Non si deve utilizzare ADCMD_PERVOL in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Questo parametro contiene una maschera nella quale i bit a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha

avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task, e il parametro io_Unit li indica come accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Leggendo il contenuto del parametro io_Unit, il task può rilevare quali sono i canali per cui l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali risultati disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare io_Command con il comando ADCMD_PERVOL. Inizializzare inoltre i seguenti parametri:

- io_Flags. Si deve inizializzare questo parametro a 0, se non si desidera utilizzarlo, altrimenti a IOF_QUICK per il QuickIO (che ha sempre successo con questo comando). Inizializzare io_Flags a ADIOF_SYNCCYCLE se si desidera che ADCMD_PERVOL cambi il periodo e il volume alla fine del ciclo in corso; in caso contrario il periodo e il volume vengono cambiati immediatamente.

- ioa_Period. Si deve inizializzare questo parametro con il nuovo valore del periodo di campionamento previsto per la riproduzione della forma d'onda. Il periodo di campionamento viene misurato in tick di sistema (un tick di sistema equivale a due cicli di clock, e quindi corrisponde a un tempo di 279,365 ns nei sistemi americani e 281,932 ns nei sistemi europei), e rappresenta il periodo di tempo che deve trascorrere fra la riproduzione di un dato e il successivo. Il suo valore può variare tra 124 e 65.535. Il filtro per l'eliminazione delle armoniche non desiderate lavora con valori inferiori a 300 o a 500, a seconda del tipo di forma d'onda.

- ioa_Volume. Si deve inizializzare questo parametro con il nuovo valore del volume per il canale indicato. Il volume può variare tra 0 (attenuazione massima) e 64 (attenuazione minima).

Discussione

ADCMD_PERVOL cambia il valore del periodo e del volume per il suono in corso di riproduzione. Se non esiste alcuna operazione di scrittura in atto nei canali specificati, ADCMD_PERVOL non ha alcun effetto.

Modificando il periodo e il volume durante la riproduzione di una forma d'onda si ottiene rispettivamente la modulazione in frequenza (effetto vibrato) e la modulazione in ampiezza (effetto tremolo).

ADCMD_SETPREC

Scopo del comando

ADCMD_SETPREC è un comando in grado di agire su più canali audio contemporaneamente, quelli indicati dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati al task e il parametro io_Unit li indica come accessibili. ADCMD_SETPREC cambia per ognuno di essi la priorità con il nuovo valore indicato dal task.

Il comando ADCMD_SETPREC viene sempre trattato in modo sincrono e la struttura di I/O viene restituita nella coda alla reply port del task solo se il flag IOF_QUICK del parametro io_Flags è stato azzerato. In questo caso, se il task è entrato in attesa tramite la funzione Wait, viene avvertito con un segnale e riottiene il controllo. Non si deve utilizzare ADCMD_SETPREC in nessun codice di interrupt a qualsiasi livello.

I risultati prodotti da questo comando sono i seguenti:

- io_Unit. Questo parametro contiene una maschera nella quale i bit a 1 indicano i canali per i quali l'operazione ha avuto successo. I bit 0-3 corrispondono ai canali 0-3, mentre gli altri sono inutilizzati.

- io_Error. Il codice restituito in questo parametro indica se la richiesta ha avuto successo. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a canali correttamente assegnati e il parametro io_Unit li indica come accessibili, il dispositivo restituisce il codice d'errore 0. Se anche una sola di queste due condizioni viene meno, il dispositivo restituisce il codice d'errore ADIOERR_NOALLOCATION. Leggendo il contenuto del parametro io_Unit, il task può rilevare quali sono i canali per cui l'operazione non ha avuto successo. Si noti che nonostante venga restituito il codice d'errore ADIOERR_NOALLOCATION, sui canali risultati disponibili il comando è stato eseguito correttamente.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa ai canali audio sui quali intende agire; memorizzare nel parametro ln_Pri la nuova priorità che intende attribuire ai canali indicati; impostare a 1 i bit del parametro io_Unit corrispondenti ai canali sui quali intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera il QuickIO (che per questo comando ha sempre successo) altrimenti impostare il parametro io_Flags a zero; impostare infine io_Command con il comando ADCMD_SETPREC.

Discussione

I canali assegnati possono avere tutti priorità distinte, stabilite durante le esecuzioni di OpenDevice, ADCMD_ALLOCATE oppure ADCMD_SETPREC. Il comando ADCMD_SETPREC viene utilizzato per stabilire in modo preciso la priorità dei canali posseduti da un task.

Un esempio d'impiego del comando ADCMD_SETPREC riguarda la diversa importanza che un task potrebbe attribuire alle tre fasi di produzione di un suono: tempo di attacco, di decadimento e di rilassamento. Il task può ritenere molto importanti le prime due fasi, dopo le quali l'ampiezza della forma d'onda si assesta per un certo periodo al livello di sostentamento, e quindi avviare la riproduzione del suono indicando un'elevata precedenza d'assegnazione. Alla fase di rilassamento, invece, il task può attribuire un'importanza molto minore e quindi chiamare ADCMD_SETPREC per ridurre a un valore inferiore la priorità d'assegnazione; in questo modo durante la fase di rilassamento altri task hanno la possibilità di sottrarre il canale e avviare la riproduzione di suoni più urgenti.

ADCMD_WAITCYCLE

Scopo del comando

ADCMD_WAITCYCLE è un comando in grado di agire su un solo canale audio alla volta, quello indicato dal task nel parametro io_Unit. Se la chiave d'assegnazione (ioa_AllocKey) corrisponde a un canale correttamente assegnato, il parametro io_Unit lo indica come accessibile e in quel canale è in corso

la riproduzione di una forma d'onda, ADCMD_WAITCYCLE restituisce la struttura di I/O al task che l'ha inviato solo quando termina il ciclo in atto nel canale indicato.

ADCMD_WAITCYCLE viene trattato in modo sincrono se una delle suddette condizioni non è verificata (codice d'errore ADIOERR_NOALLOCATION), oppure se nell'unità indicata non è in corso alcuna riproduzione. In questi casi, il task riceve alla sua reply port la struttura di I/O solo se non ha impostato il flag IOF_QUICK del parametro io_Flags.

In tutti gli altri casi ADCMD_WAITCYCLE viene trattato in modo asincrono: il dispositivo azzera il flag IOF_QUICK e la struttura di I/O viene restituita al task solo a completamento del ciclo. Questo comando permette a un task di sincronizzarsi con i cicli di una forma d'onda riprodotta da uno dei quattro canali.

Non si deve utilizzare ADCMD_WAITCYCLE in codici di interrupt che lavorano con livelli di interrupt pari o superiori a 5.

I risultati prodotti da questo comando sono i seguenti:

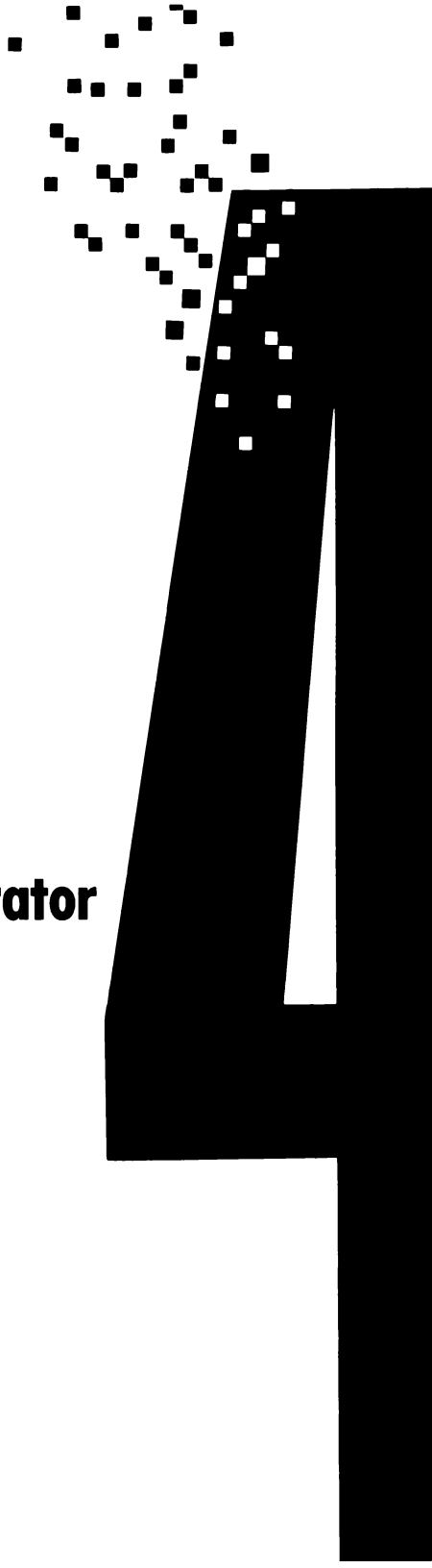
- io_Unit. È una mappa di 4 bit che rappresenta il canale per il quale il task attende la fine del ciclo in atto; i bit 0-3 corrispondono ai canali 0-3.

- io_Flags. Il flag IOF_QUICK viene azzerato se esiste un comando CMD_WRITE in esecuzione nel canale selezionato.

- io_Error. Un valore di questo parametro pari a 0 indica che il comando è stato eseguito. Il codice d'errore ADIOERR_NOALLOCATION indica che il parametro ioa_AllocKey della struttura IOAudio non contiene l'attuale chiave d'assegnazione, oppure che l'unità indicata da io_Unit non è disponibile. Il codice d'errore ADIOERR_ABORTED indica che il comando ADCMD_WAITCYCLE è stato eliminato.

Preparazione della struttura IOAudio

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il task deve inizializzare il parametro ioa_AllocKey con la chiave d'assegnazione relativa al canale audio sul quale intende agire; impostare a 1 il bit del parametro io_Unit corrispondente al canale sul quale intende agire; impostare il flag IOF_QUICK del parametro io_Flags se desidera che in caso d'insuccesso non venga restituita alcuna struttura di I/O; inizializzare infine io_Command con il comando ADCMD_WAITCYCLE.



Il dispositivo Narrator

Introduzione

Questo capitolo illustra le funzioni e i comandi del dispositivo Narrator, e la funzione Translate della libreria Translator. Apprenderemo così le procedure che consentono di far elaborare all'Amiga file di testo, di convertirli in stringhe di fonemi e di riprodurli attraverso gli altoparlanti. In questo modo si riesce a far "parlare" l'Amiga, cioè a fargli leggere e pronunciare qualsiasi testo. L'Amiga effettua la conversione in fonemi presupponendo che il testo sia scritto in inglese e quindi se il testo è in italiano si ottiene una pronuncia imperfetta. Per ovviare all'inconveniente occorre che i task provvedano autonomamente alla traduzione in fonemi della stringa di testo.

Questo dispositivo può essere utile in molteplici applicazioni, dai programmi che prevedono un'interfaccia vocale oltre a quella grafica, ai programmi per le video-vetrine, alle applicazioni che si rivolgono ai non vedenti.

Il dispositivo Narrator viene programmato impiegando alcune funzioni della libreria Exec e sei comandi standard. In questo capitolo vengono inoltre analizzati gli importanti legami esistenti fra Narrator e Audio.

Elaborazione del testo

La Figura 4.1 (nella pagina successiva) mostra come avviene l'elaborazione di una stringa di testo attraverso la libreria Translator, il dispositivo Narrator e le routine interne del dispositivo Audio. Questa figura è una guida per creare task adibiti alla lettura di testi, per esempio da inserire in un word processor evoluto.

Il task memorizza la stringa di testo in un suo buffer che ha appositamente allocato in memoria. La libreria Translator, utilizzando la libreria interna Voice Synthesis, traduce parola per parola la stringa di testo in una stringa equivalente di fonemi. Per effettuare questa traduzione, la libreria Translator consulta anche una tavola di eccezioni contenuta al suo interno. Le parole che compaiono in questa tavola hanno una "traduzione" predefinita, mentre le altre vengono tradotte in fonemi ricorrendo a una serie di regole. Questa traduzione automatica offre buoni risultati solo per i testi in lingua inglese. Se la lingua è per esempio l'italiano, le strade da seguire sono due: scrivere una nuova libreria Translator "italiana" o creare un programma in grado di tradurre autonomamente le parole italiane nei corretti fonemi. In ogni caso il risultato dev'essere sempre una stringa di fonemi memorizzata in un buffer.

A questo punto il dispositivo Narrator può istruire le routine interne del dispositivo Audio perché riproducano la stringa di fonemi utilizzando i dati della struttura narrator_rb; questa operazione viene definita "di scrittura" (CMD_WRITE). Il dispositivo Narrator fornisce anche una serie di parametri che descrivono la sagoma di una bocca; questi parametri vengono memorizzati

nella struttura mouth_rb, che il task può impiegare per visualizzare sullo schermo l'animazione di una bocca che si muove in modo diverso a seconda del fonema che sta pronunciando; questa operazione viene definita "di lettura" (CMD_READ).

comandi del dispositivo Narrator

La Tavola 4.1 (nella pagina successiva) offre un sommario elenco dei comandi previsti dal dispositivo Narrator. Dal momento che questo dispositivo non prevede alcun comando specifico, nella tavola compaiono solo comandi standard. Due di essi prevedono modifiche delle strutture relative: CMD_READ e CMD_WRITE, gli stessi che prevedono la restituzione di un codice d'errore nel parametro io_Error della sotto-struttura IOStdReq contenuta nella struttura narrator_rb. Gli altri comandi previsti dal dispositivo non alterano nessun parametro delle strutture di I/O che li descrivono e azzerano sempre il parametro io_Error. Nessun comando del dispositivo Narrator può essere inoltrato chiedendo il QuickIO; i comandi CMD_READ e CMD_WRITE possono essere accodati alla request port del dispositivo, mentre gli altri vengono sempre eseguiti in modo immediato.

L'invio dei comandi al dispositivo Narrator

La Figura 4.2a (a pagina 142) mostra lo schema generale per inviare comandi alle routine interne del dispositivo Narrator. Le linee con le frecce rappresentano i parametri che i task devono inizializzare. Nella Figura 4.2b (a pagina 143) le frecce rappresentano invece i parametri restituiti dalle routine interne del dispositivo.

La prima fase, sulla quale il programmatore possiede un completo controllo, è costituita dalla preparazione della struttura di I/O che descrive il comando da inviare. La scelta dei parametri da inizializzare dipende dal

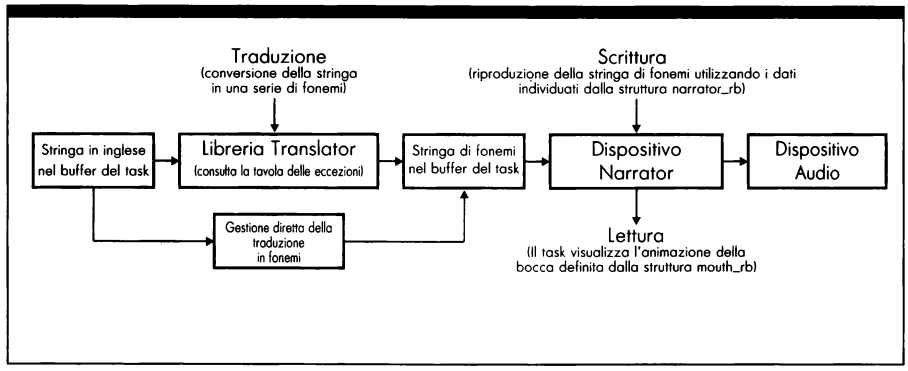


Figura 4.1: *Elaborazione di una stringa in inglese*

Comando	QuickIO possibile?	QueuedIO possibile?	Parametri influenzati	io_Error
CMD_FLUSH	No	No	-	Impostato a 0
CMD_READ	No	Sì	larghezza altezza bocca	Impostato al codice d'errore
CMD_RESET	No	No	-	Impostato a 0
CMD_START	No	No	-	Impostato a 0
CMD_STOP	No	No	-	Impostato a 0
CMD_WRITE	No	Sì	io_Actual	Impostato al codice d'errore

Tavola 4.1: *Comandi previsti dal dispositivo Narrator*

comando. Quando si apre il dispositivo con una chiamata alla funzione OpenDevice, i parametri della voce presenti nella struttura narrator_rb vengono inizializzati con valori di default. Se il task desidera personalizzare le caratteristiche della voce, deve modificare quei parametri prima d'inviare il comando CMD_WRITE.

La seconda fase prevede l'invio della richiesta e la sua elaborazione da parte delle routine interne del dispositivo Narrator. Il task invia la richiesta di I/O utilizzando i comandi BeginIO, DoIO o SendIO. In particolare, il task invia generalmente i comandi CMD_WRITE in modo asincrono (BeginIO e SendIO) per poter poi inviare in modo sincrono (DoIO) diversi comandi CMD_READ e ottenere la sequenza di sagome della bocca.

A questo punto la richiesta viene accodata (comandi CMD_WRITE e CMD_READ), oppure viene elaborata immediatamente (tutti gli altri comandi). L'elaborazione viene condotta da un pool di routine comprendente le routine dei dispositivi Narrator, Audio e della libreria Exec.

La terza fase prevede l'elaborazione dei parametri da restituire al task e la restituzione della struttura di I/O (che nel caso di comandi non immediati viene sempre restituita dal momento che il dispositivo Narrator non prevede il QuickIO).

I risultati prodotti dall'elaborazione del comando sono i parametri mostrati nella Figura 4.2b (a pagina 143). Il parametro io_Error può contenere uno dei 15 codici d'errore previsti dal dispositivo Narrator per i comandi CMD_WRITE

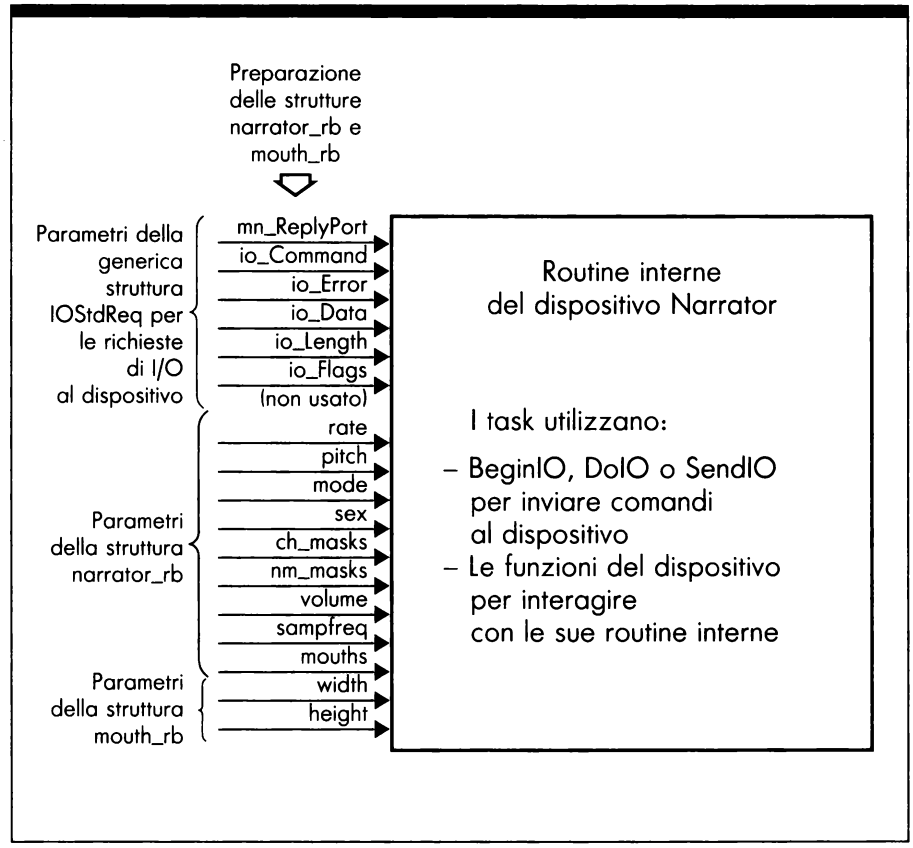


Figura 4.2a: *Gestione delle funzioni e dei comandi previsti dal dispositivo Narrator (input)*

e CMD_READ. La Figura 4.2b (nella pagina successiva) mostra anche i parametri che il dispositivo inizializza e restituisce dopo l'esecuzione della funzione OpenDevice (io_Device, io_Unit, lib_OpenCnt, unit_OpenCnt e tutti i parametri della struttura narrator_rb che caratterizzano la voce), nonché dopo l'elaborazione del comando CMD_READ (i parametri della struttura mouth_rb).

Le strutture del dispositivo Narrator

Come si nota osservando la Figura 4.3 (a pagina 144), il dispositivo Narrator prevede due strutture di I/O non standard: narrator_rb e mouth_rb. Entrambe sono messaggi che hanno come primo parametro la struttura di I/O standard IOStdReq. La struttura narrator_rb descrive le caratteristiche della voce che il dispositivo deve impiegare quando pronuncia la stringa di fonemi che il task ottiene chiamando la funzione Translate o che crea con altri sistemi. Questa è la struttura che il task deve usare per tutti i comandi, tranne che per CMD_READ. Per quest'ultimo, infatti, il dispositivo prevede la struttura

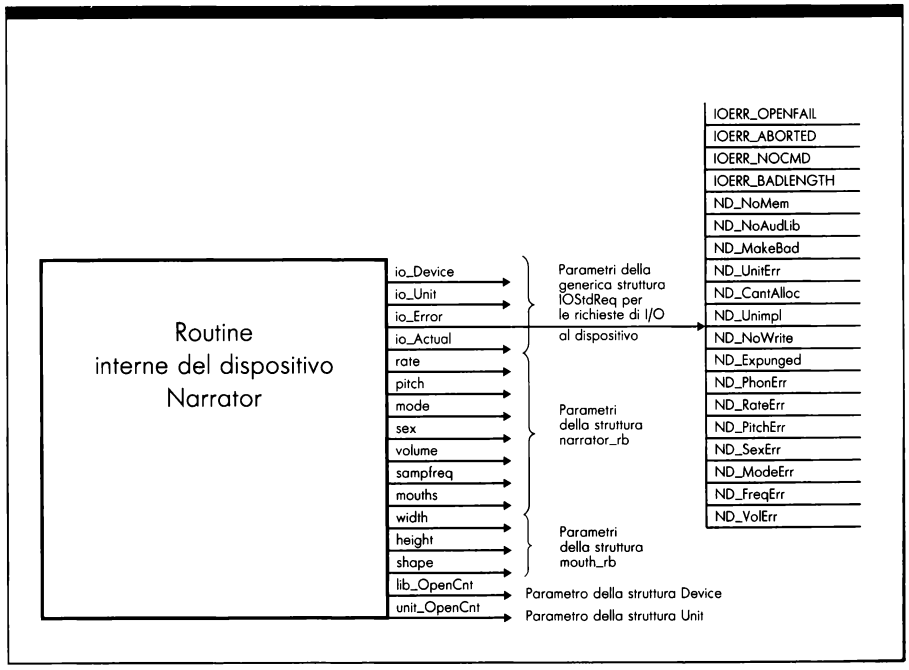


Figura 4.2b: *Gestione dei comandi e delle funzioni previsti dal dispositivo Narrator (output)*

mouth_rb, un'estensione della struttura narrator_rb. Rispetto alla prima. mouth_rb aggiunge una serie di tre parametri nei quali il dispositivo memorizza i dati relativi alla nuova sagoma della bocca; dal momento che la sagoma restituita da CMD_READ corrisponde al fonema che il dispositivo sta pronunciando, il task può utilizzarne i dati per animare sullo schermo una bocca mentre la macchina articola i suoni. Le strutture narrator_rb e mouth_rb non sono direttamente collegate tra loro. La libreria Translator non prevede l'uso di particolari strutture da parte del programmatore.

La struttura narrator_rb

La struttura narrator_rb contiene una sotto-struttura IOStdReq denominata message, che viene utilizzata per intestare il messaggio e definirne alcuni parametri standard (come io_Device, io_Unit, io_Command, io_Flags, io_Data...). In particolare, in questa sotto-struttura il puntatore mn_ReplyPort deve sempre indicare il mittente della richiesta, cioè la struttura MsgPort che rappresenta la coda alla reply port del task che ha inviato il comando; a questa message port vengono restituite le richieste di I/O a mano a mano che il dispositivo le elabora.

Oltre alla sotto-struttura message, la struttura di I/O narrator_rb contiene al suo interno una serie di parametri specifici del dispositivo Narrator, tramite i quali i task descrivono le caratteristiche della voce da impiegare nella

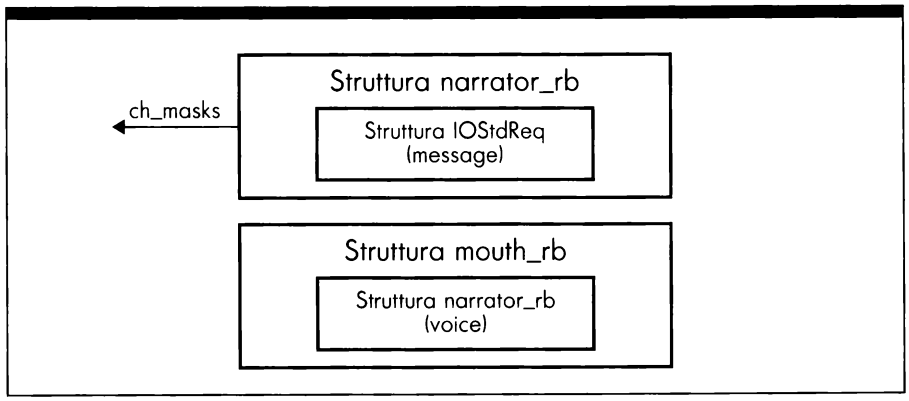


Figura 4.3: *Strutture utilizzate dal dispositivo Narrator*

pronuncia dei fonemi. Fra i suoi parametri riveste particolare importanza ch_masks, che il task deve aggiornare con l'indirizzo di un array di combinazioni dei canali, e nm_masks, che il task deve aggiornare con il numero di byte contenuti nell'array. In quest'array, ampiamente descritto nel capitolo precedente, i task devono indicare le combinazioni di canali audio che il dispositivo Narrator può tentare di ottenere tramite il dispositivo Audio; attraverso i canali della combinazione che ha avuto successo, il dispositivo Audio (sotto la supervisione del dispositivo Narrator) produce i suoni che costituiscono la voce. Si noti che il dispositivo Narrator provvede ad aprire subito il dispositivo Audio, ma richiede l'accesso ai canali audio solo quando elabora i comandi CMD_WRITE. Allo stesso modo, libera sempre i canali non appena termina l'elaborazione di un comando CMD_WRITE (la riproduzione vocale di un testo), ma chiude il dispositivo Audio solo quando il task manda in esecuzione la funzione CloseDevice. Per quanto riguarda le priorità nell'uso dei canali, valgono le considerazioni fatte nel precedente capitolo, tenendo conto che il dispositivo Narrator apre le combinazioni di canali audio con priorità 75 (i task non possono cambiarla). Per maggiori dettagli vedere la scheda del comando CMD_WRITE. Inoltre, il dispositivo Narrator può essere aperto da diversi task contemporaneamente, che si trovano quindi a condividere le sue risorse. Prescindendo dalla combinazione di canali che per ogni task il dispositivo riesce ad assegnare, se un task invia un comando CMD_WRITE per avviare la riproduzione vocale di un testo, gli eventuali comandi CMD_WRITE che gli altri task inviano subito dopo vengono semplicemente accodati ed elaborati quando arriva il loro turno. In altre parole, anche se il dispositivo Narrator è teoricamente in grado di gestire le esigenze di più task, in realtà può riprodurre solo un testo alla volta.

La struttura narrator_rb è definita come segue:

```
struct narrator_rb {
    struct IOStdReq message;
    UWORD rate;
    UWORD pitch;
```

```
    UWORD mode;
    UWORD sex;
    UBYTE *ch_masks;
    UWORD nm_masks;
    UWORD volume;
    UWORD sampfreq;
    UBYTE mouths;
    UBYTE chanmask;
    UBYTE numchan;
    UBYTE pad;
};
```

I parametri della struttura narrator_rb hanno i seguenti significati:

- message. È il nome della sotto-struttura di tipo IOStdReq contenuta nella struttura narrator_rb. Il parametro mn_ReplyPort di questa sotto-struttura dev'essere inizializzato dal task con l'indirizzo della struttura MsgPort che rappresenta la coda alla sua reply port.

- rate. Rappresenta la velocità con cui vengono articolate le parole. Può variare tra 40 e 400 parole al minuto. Quando si chiama la funzione OpenDevice, il dispositivo memorizza in questo parametro il valore di default 150; il task può comunque variarlo come preferisce.

- pitch. Rappresenta la frequenza della voce. Si tratta di un valore medio attorno al quale oscilla la frequenza della voce se è stata scelta una voce dotata di espressività. Può variare tra 65 e 320 Hz; OpenDevice memorizza in questo parametro il valore di default 110.

- mode. Determina il tipo di voce. Si può scegliere fra una voce monotona e priva di espressività ("robotica"), e una voce dotata di maggior naturalezza. OpenDevice memorizza in questo parametro la costante NATURALF0, corrispondente alla voce con intonazione naturale, mentre la voce robotica viene attivata memorizzando nel parametro mode la costante ROBOTICF0.

- sex. Indica il sesso della voce. OpenDevice memorizza in questo parametro il valore di default MALE (maschile), ma il task può cambiarlo eventualmente in FEMALE (femminile).

- ch_masks. In questo puntatore il task deve memorizzare l'indirizzo di un array di combinazioni dei canali che ha precedentemente allocato in memoria. Il dispositivo Narrator impiega questo array per accedere ai canali del dispositivo Audio quando esegue un comando CMD_WRITE. Ogni byte di quest'array deve contenere nel nibble meno significativo un valore compreso fra 1 e 15, che rappresenta una delle 15 combinazioni di canali audio previste dall'Amiga. Per maggiori dettagli sull'array di combinazioni dei canali si consulti il capitolo precedente.

- nm_masks. In questo parametro il task deve memorizzare il numero di byte contenuti nell'array. Ogni byte corrisponde a una combinazione di canali.

- volume. Controlla il volume della voce, e può contenere un valore compreso fra 0 (attenuazione massima del segnale) e 64 (attenuazione minima del segnale). OpenDevice memorizza in questo parametro il valore di default 64, ossia il volume massimo.

- sampfreq. Indica la fequenza di campionamento utilizzata per riprodurre i dati corrispondenti ai fonemi contenuti nel testo da riprodurre, frequenza che è espressa in dati-campione al secondo. La frequenza di campionamento può variare tra 5.000 e 28.000 dati-campione al secondo. OpenDevice memorizza in questo parametro il valore di default 22.200.

- mouths. Dev'essere inizializzato a 1 se il task, inviando un comando CMD_WRITE, desidera che il dispositivo calcoli una nuova sagoma della bocca ogni volta che riceve il comando CMD_READ. Le sagome vengono generate solo durante l'emissione di fonemi e simulano il suono che il dispositivo Audio sta riproducendo. Se il task non desidera ricevere i dati relativi alle espressioni della bocca, deve impostare il parametro mouths a 0. Il suo valore di default è zero.

- chanmask. Viene aggiornato dal dispositivo con la combinazione di canali audio che è riuscito ad assegnare durante l'elaborazione del comando CMD_WRITE. Si tratta ovviamente di una delle combinazioni indicate nell'array puntato dal parametro ch_masks. Si noti che questo parametro non viene azzerato quando al termine della riproduzione il comando CMD_WRITE libera la combinazione di canali. Il suo impiego è riservato al dispositivo.

- numchan. Viene aggiornato dal dispositivo con il numero di canali audio indicati nella combinazione che ha avuto successo durante l'elaborazione del comando CMD_WRITE. Si noti che questo parametro non viene azzerato quando al termine della riproduzione il comando CMD_WRITE libera la combinazione di canali. Il suo impiego è riservato al dispositivo.

- pad. Si tratta di un byte che viene impiegato solo per allineare in memoria la fine della struttura narrator_rb alle word.

La struttura mouth_rb

La struttura mouth_rb contiene una sotto-struttura narrator_rb (di nome voice) e quindi ne costituisce un'estensione. Anch'essa costituisce un messaggio e il task deve usarla per inviare il comando CMD_READ. Ogni volta

che il dispositivo riceve questo comando ed è in corso una riproduzione vocale, restituisce nei parametri addizionali della struttura mouth_rb dettagliate informazioni sulla sagoma della bocca in relazione alla sillaba pronunciata.

Generalmente il task alloca una struttura narrator_rb, che viene inizializzata tramite una chiamata a OpenDevice, e una struttura mouth_rb nella cui sotto-struttura voice copia l'intero contenuto della struttura narrator_rb. Per le due strutture il task può anche decidere di allocare e impiegare due diverse reply port, in modo che in una arrivino le risposte relative ai comandi CMD_WRITE e nell'altra le strutture mouth_rb relative ai comandi CMD_READ.

La struttura mouth_rb è definita come segue:

```
struct mouth_rb {
    struct narrator_rb voice;
    UBYTE width;
    UBYTE height;
    UBYTE shape;
    UBYTE pad;
};
```

I parametri della struttura mouth_rb hanno il seguente significato:

- voice. È il nome della sotto-struttura di tipo narrator_rb che costituisce il messaggio base previsto dal dispositivo. I parametri di questa sotto-struttura dovrebbero essere copiati dalla struttura narrator_rb impiegata per aprire il dispositivo.

- width. Contiene un numero intero fornito dalle routine interne del dispositivo Narrator che rappresenta la larghezza della bocca. Questo parametro cambia ogni volta che il dispositivo risponde a un comando CMD_READ e simula la larghezza che una bocca reale assumerebbe pronunciando la sillaba. In questo modo ogni sagoma è sempre diversa dalla precedente e l'eventuale animazione della bocca che il task può realizzare risulta particolarmente realistica. I valori che questo parametro può assumere variano da 0 a 14.

- height. Contiene un numero intero fornito dalle routine interne del dispositivo Narrator che rappresenta l'altezza della bocca. Questo parametro cambia ogni volta che il dispositivo risponde a un comando CMD_READ e simula l'altezza che una bocca reale assumerebbe pronunciando la sillaba. In questo modo ogni sagoma è sempre diversa dalla precedente e l'eventuale animazione della bocca che il task può realizzare risulta particolarmente realistica. I valori che questo parametro può assumere variano da 0 a 14.

- shape. Contiene la definizione interna della sagoma della bocca fornita dalle routine interne del dispositivo Narrator. Questo parametro è di esclusiva competenza del dispositivo e cambia ogni volta che il dispositivo risponde a un comando CMD_READ.

- pad. Si tratta di un byte che viene impiegato solo per allineare in memoria la fine della struttura mouth_rb alle word.

Codici d'errore del dispositivo Narrator

I codici d'errore restituiti dal dispositivo Narrator hanno i seguenti significati:

- ND_NoMem. Nel sistema non è disponibile memoria sufficiente per allocare i dati e le strutture necessarie alle routine interne del dispositivo Narrator. Viene restituito solo durante l'apertura del dispositivo effettuata tramite la funzione OpenDevice.

- ND_NoAudLib. Il sistema non ha trovato la libreria del dispositivo Audio. Nell'Amiga 1000 il dispositivo Audio viene caricato nella ROM WCS (Write Control Store) durante l'attivazione della macchina. Questo tipo d'errore non può verificarsi negli Amiga 500 e Amiga 2000, a meno che le ROM contenenti il sistema operativo non presentino malfunzionamenti. Questo codice d'errore viene restituito solo dalla funzione OpenDevice.

- ND_MakeBad. Si è verificato un errore durante una chiamata alla funzione MakeLibrary, il quale a sua volta ha causato un cattivo funzionamento della libreria del dispositivo Narrator o Audio.

- ND_UnitErr. Negli argomenti della funzione OpenDevice il task ha indicato un'unità del dispositivo Narrator diversa dall'unica consentita (l'unità 0).

- ND_CantAlloc. Le routine interne del dispositivo Narrator non sono riuscite ad assegnare nessuna combinazione di canali tra quelle indicate nell'array puntato da ch_masks. In genere questo errore viene restituito dal comando CMD_WRITE quando in ogni combinazione di canali indicata dal task almeno un canale è assegnato a un altro task con una priorità d'assegnazione superiore a 75.

- ND_Unimpl. Il task ha indicato nel parametro io_Command della richiesta di I/O un codice di comando non previsto dal dispositivo (per esempio, CMD_INVALID).

- ND_NoWrite. Il task ha inviato un comando CMD_READ senza prima aver inviato un comando CMD_WRITE, oppure è terminata l'esecuzione di un comando CMD_WRITE e quindi il dispositivo non restituisce nessuna nuova sagoma nella struttura mouth_rb impiegata per inviare il comando CMD_READ.

- ND_Expunged. Il sistema non può aprire la libreria del dispositivo Narrator o del dispositivo Audio perché il flag LIBF_DELEXP nel parametro lib_Flags delle rispettive strutture Library risulta impostato.

- ND_PhonErr. Si è verificato un errore di riconoscimento nella stringa dei fonemi durante l'esecuzione di un comando CMD_WRITE. Le routine interne del dispositivo Narrator leggendo i fonemi della stringa ne hanno incontrato uno del quale non comprendono la sintassi.

- ND_RateErr. La velocità di riproduzione richiesta dal task non rientra nei limiti consentiti (40-400 parole al minuto).

- ND_PitchErr. La frequenza media della voce richiesta dal task per la riproduzione vocale non rientra nei limiti consentiti (65-320 Hz).

- ND_SexErr. Il parametro sex della struttura narrator_rb non è stato inizializzato correttamente (gli unici valori possibili sono MALE e FEMALE).

- ND_ModeErr. Il modo di riproduzione richiesto non è consentito (gli unici valori possibili sono NATURALF0 e ROBOTICF0).

- ND_FreqErr. La frequenza di campionamento indicata dal task non rientra nei limiti consentiti (5.000-28.000 dati al secondo).

- ND_VolErr. Il volume indicato dal task per la voce non rientra nei limiti consentiti (0-64).

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (narrator_rb)
A1

Scopo della funzione

Questa funzione chiude l'accesso da parte del task all'unica unità del dispositivo Narrator, l'unità 0. La funzione CloseDevice per il dispositivo Narrator prevede la chiusura del dispositivo Audio (che viene aperto automaticamente nel momento in cui il task chiama OpenDevice per aprire il dispositivo Narrator) e la decrementazione del parametro lib_OpenCnt contenuto nella struttura Device del dispositivo. Se in questa fase lib_OpenCnt viene a contenere il valore zero e il flag LIBF_DELEXP del parametro lib_Flags nella struttura Device risulta impostato, il dispositivo Narrator viene eliminato dalla memoria.

La funzione CloseDevice provvede inoltre ad aggiornare con il valore −1 il puntatore io_Device della sotto-struttura IOStdReq presente nella struttura narrator_rb, e con il valore 0 il puntatore io_Unit, che per il dispositivo Narrator svolge un compito diverso da quello descritto nei capitoli introduttivi. Per maggiori dettagli sul significato del parametro io_Unit si veda la descrizione della funzione OpenDevice.

Argomenti della funzione

narrator_rb Deve contenere l'indirizzo della struttura di I/O narrator_rb che il task impiega per interagire con il dispositivo Narrator. Generalmente questa struttura è la stessa che viene parzialmente inizializzata dalla funzione OpenDevice quando il task apre il dispositivo.

Discussione

CloseDevice permette a un task di concludere l'accesso all'unità 0 del dispositivo Narrator. Dato che si tratta dell'unica unità di cui il dispositivo dispone, CloseDevice chiude sempre l'accesso all'intero dispositivo Narrator da parte del task. Dato che il dispositivo Narrator può essere condiviso, il suo impiego si deve ritenere realmente concluso soltanto quando tutti i task che l'hanno aperto hanno chiamato la funzione CloseDevice, cioè quando il parametro lib_OpenCnt della struttura Device contiene il valore zero.

CloseLibrary

Sintassi di chiamata della funzione

CloseLibrary (library)
A1

Scopo della funzione

Questa funzione della libreria Exec chiude l'accesso da parte del task alla libreria indicata come argomento. Nel caso del dispositivo Narrator, il task può trovarsi nella necessità di chiamare la funzione Translate per tradurre in fonemi una stringa di testo, e questa funzione si trova nella libreria Translator (translator.library) che il task deve quindi provvedere ad aprire tramite la funzione OpenLibrary e chiudere tramite la funzione CloseLibrary. CloseLibrary decrementa di 1 il parametro lib_OpenCnt della struttura Library che definisce in memoria la libreria, così da indicare la diminuzione del numero dei task che hanno accesso alla libreria.

Se il parametro lib_OpenCnt arriva a 0, e c'è una richiesta di eliminazione della libreria precedentemente prorogata (ovvero risulta impostato il flag LIBF_DELEXP del parametro lib_Flags), le routine interne della libreria vengono eliminate dalla RAM non appena CloseLibrary restituisce il controllo al task.

Una volta che il task ha provveduto a chiudere la libreria Translator tramite CloseLibrary, non può più accedere alla funzione Translate senza ripetere l'intera operazione.

Argomenti della funzione

library — Questo argomento dev'essere l'indirizzo della struttura Library relativa alla libreria che il task ottiene chiamando la funzione OpenLibrary.

Discussione

Nel caso del dispositivo Narrator, la funzione CloseLibrary viene impiegata per concludere l'accesso alla libreria Translator, aperta dal task per chiamare la funzione Translate. Ogni task che utilizza la libreria Translator deve

chiamare la funzione CloseLibrary quando non ne ha più bisogno; le chiamate alle funzioni OpenLibrary e CloseLibrary devono essere sempre accoppiate. Si ricordi che la funzione Translate è l'unica funzione della libreria Translator disponibile ai task.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice("narrator.device", ØL, narrator_rb, ØL)
DØ                 AØ                  DØ A1          D1
```

Scopo della funzione

Questa funzione permette al task di accedere all'unità 0 del dispositivo Narrator. Il task deve indicare come argomento l'indirizzo di una struttura di tipo narrator_rb che ha allocato in memoria, nella quale OpenDevice inizializza i parametri standard, come io_Device e io_Unit, e alcuni parametri tipici del dispositivo Narrator. Questi ultimi sono quelli che descrivono le caratteristiche della voce che il dispositivo deve generare quando riceve i comandi CMD_WRITE; la funzione OpenDevice li aggiorna con i loro valori di default.

Il dispositivo Narrator è ad accesso condiviso e non può essere in nessun caso aperto in modo esclusivo. Se al momento della chiamata a OpenDevice il dispositivo non è presente in memoria, il sistema, in maniera trasparente ai task, provvede a caricarlo da disco e ad allocarlo. La prima volta che il dispositivo viene aperto provvede ad aprire anche il dispositivo Audio (e lo chiude automaticamente al momento della sua chiusura definitiva). Si noti che OpenDevice apre automaticamente il dispositivo Audio ma non assegna nessuna combinazione di canali; questa operazione viene infatti compiuta dal dispositivo solo quando riceve ed elabora un comando CMD_WRITE.

Quando OpenDevice restituisce il controllo, un valore pari a 0 nella variabile error, o nel parametro io_Error della struttura di I/O, indica che la richiesta di apertura del dispositivo Narrator ha avuto successo. In questo caso nella struttura di I/O passata come argomento OpenDevice ha inizializzato il puntatore io_Device con l'indirizzo della struttura Device che definisce il dispositivo Narrator, e il puntatore io_Unit con un particolare numero intero che il dispositivo Narrator impiega internamente. Si tratta di un numero che viene incrementato di 1 ogni volta che un task esegue una chiamata alla funzione OpenDevice; il dispositivo lo utilizza per stabilire se due comandi CMD_WRITE e CMD_READ provengono dallo stesso task oppure no. Per certi versi, questo numero svolge un compito simile a quello della chiave d'assegnazione vista nel

precedente capitolo. Il task deve indicarlo nei parametri io_Unit di tutte le strutture di I/O che intende inviare al dispositivo, al pari dell'indirizzo contenuto nel parametro io_Device. Si noti che il parametro io_Unit svolge un compito completamente diverso da quello descritto nei primi due capitoli: anche in questo caso, come per il dispositivo Audio, si tratta di un'eccezione.

OpenDevice si preoccupa anche d'incrementare il parametro lib_OpenCnt della struttura Device del dispositivo.

Se l'apertura del dispositivo non ha avuto successo, la funzione OpenDevice restituisce uno dei seguenti codici d'errore.

- IOERR_OPENFAIL. Non è stato possibile aprire il dispositivo Narrator (per esempio perché non si trova su disco).

- ND_NoAudLib. Non è stato possibile aprire il dispositivo Audio in quanto non risulta disponibile la relativa libreria.

- ND_NoMem. Il sistema non ha potuto allocare la necessaria memoria per aprire simultaneamente i dispositivi Narrator e Audio.

- ND_UnitErr. Il task ha indicato un numero di unità diverso da 0.

Quelli che seguono sono i parametri aggiornati restituiti dalla funzione OpenDevice:

- io_Device. Viene inizializzato con l'indirizzo della struttura di tipo Device di gestione del dispositivo Narrator. La struttura Device è unica per tutti i task e contiene le informazioni necessarie per accedere ai dati e alle routine contenute nella libreria del dispositivo.

- io_Unit. Viene inizializzato con un numero intero che il dispositivo associa al task, per poter distinguere le sue richieste di I/O da quelle di altri task. Il task deve copiare questo valore in tutte le strutture di I/O che intende usare con il dispositivo Narrator.

- io_Error. Il dispositivo restituisce in questo parametro uno dei codici d'errore appena illustrati.

Oltre ai precedenti parametri (comuni a tutti i dispositivi), la funzione OpenDevice inizializza con i loro valori di default alcuni parametri caratteristici della struttura narrator_rb.

- rate 150 parole al minuto.

- pitch 110 Hz.

- mode NATURALF0.

- sex MALE.

- mouths — 0 (ordina al dispositivo di non generare le sagome della bocca).
- sampfreq — 22.200 dati-campione al secondo.
- volume — 64, il valore massimo.

Argomenti della funzione

"narrator.device" — Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Narrator.

ØL — Rappresenta il numero dell'unità a cui si desidera accedere (il dispositivo Narrator ne prevede solo una, individuata dal numero 0).

narrator_rb — Rappresenta l'indirizzo della struttura narrator_rb allocata dal task per interagire con il dispositivo.

ØL — Questo valore indica che l'argomento flag non viene preso in considerazione dal dispositivo Narrator.

Preparazione della struttura narrator_rb

Per aprire il dispositivo Narrator basta inizializzare il parametro mn_ReplyPort della struttura di I/O con l'indirizzo della struttura MsgPort che rappresenta la reply port del task. A questa message port arrivano le risposte ai comandi inviati in modo asincrono (principalmente il comando CMD_WRITE). Il task può allocare una message port tramite la funzione CreatePort di supporto alla libreria Exec e indicarne l'indirizzo come argomento della funzione CreateExtIO. Chiamando quest'ultima funzione, il task alloca la struttura di I/O necessaria per interagire con il dispositivo e automaticamente memorizza nel parametro mn_ReplyPort l'indirizzo della sua reply port.

Discussione

La funzione OpenDevice rende disponibili ai task le routine del dispositivo Narrator. Il dispositivo Narrator può essere aperto soltanto nel modo di accesso condiviso e che non consente l'esecuzione automatica di un comando tramite OpenDevice. Quando il task non ha più bisogno del dispositivo deve provvedere a chiuderlo chiamando CloseDevice, che chiude anche il dispositivo Audio se il parametro lib_OpenCnt della struttura Device è a zero.

OpenLibrary

Sintassi di chiamata della funzione

```
TranslatorBase = OpenLibrary ("translator.library", ØL)
DØ                           A1                      DØ
```

Scopo della funzione

Questa funzione standard della libreria Exec apre la libreria che il task ha indicato negli argomenti della funzione. Nel caso del dispositivo Narrator, il task impiega la funzione OpenLibrary per aprire la libreria Translator e chiamare la funzione Translate contenuta al suo interno. Il task ha bisogno di questa funzione per trasformare una o più stringhe di testo in stringhe di fonemi. OpenLibrary restituisce l'indirizzo base della libreria, che nel caso della libreria Translator dev'essere memorizzato nella variabile globale TranslatorBase affinché il linker possa risolvere correttamente il riferimento esterno. OpenLibrary incrementa il parametro lib_OpenCnt della struttura Library.

Argomenti della funzione

"translator.library" Il task deve indicare in questo argomento la stringa contenente il nome della libreria Translator.

ØL Questo argomento indica al sistema che il task accetta qualsiasi versione della libreria Translator presente su disco. Si ricordi sempre che la funzione OpenLibrary si aspetta in questo argomento una long word.

Discussione

Ogni task che desideri utilizzare la funzione Translate deve aprire la libreria Translator tramite una chiamata a OpenLibrary. Quando la libreria è stata aperta, il task può utilizzare la funzione Translate per "tradurre" una o più stringhe di testo nelle corrispondenti stringhe di fonemi e cederle quindi al dispositivo Narrator perché vengano riprodotte.

Dal linguaggio C la libreria Translator rende direttamente disponibile solo

la funzione Translate. La libreria Translator può essere aperta contemporaneamente da un numero qualsiasi di task. Ognuno di essi riceve da OpenLibrary l'indirizzo della stessa struttura Library. Solo quando tutti i task che avevano aperto la libreria provvedono a chiuderla il sistema può eliminarla dalla memoria.

Si noti che la libreria Translator risiede su disco. Quindi, perché OpenLibrary possa aprirla senza inconvenienti, occorre che nella directory logica LIBS: si trovi il file translator.library.

Translate

Sintassi di chiamata della funzione

```
return_code = Translate (stringaTesto, lunghezza_stringaTesto,
D0                       A0            D0
                         stringaFonemi, lunghezza_stringaFonemi)
                         A1             D1
```

Scopo della funzione

Questa funzione converte una stringa di testo in una stringa di fonemi, dando per scontato che il testo sia in lingua inglese. Se il task invia un testo in un'altra lingua, la funzione lo trasforma in una stringa di fonemi che produce una lettura con pronuncia inglese.

La funzione Translate restituisce il valore 0 nella variabile return_code se non si è verificato alcun errore durante il processo di traduzione del testo in fonemi. L'unico errore possibile sarebbe l'overflow del buffer in cui viene immagazzinata la stringa trasformata in sequenza di fonemi (stringaFonemi), dovuto al fatto che la trasformazione ha generato una stringa molto più lunga della stringa di testo e non prevista dal task (i fonemi possono occupare diversi caratteri). Se questo accade, Translate blocca il processo di traduzione in corrispondenza dell'ultima parola tradotta che non supera la grandezza del buffer, e restituisce nella variabile return_code l'offset alla parola della stringa di testo (stringaTesto) dalla quale dovrebbe riprendere la traduzione. Il valore assoluto di questo offset sommato all'indirizzo di stringaTesto individua il nuovo indirizzo da passare a Translate quando il task la chiama per continuare la traduzione da dove si è interrotta.

Questo sistema permette al task di racchiudere in un'unica stringa un testo molto lungo e d'impiegare un buffer stringaFonemi di modeste dimensioni; ogni volta che Translate restituisce un valore diverso da zero, il task lo *sottrae* dall'indirizzo stringaTesto per ottenere un nuovo indirizzo e continua a

chiamare Translate fino a quando questa funzione non restituisce il valore zero.

Quando Translate restituisce il controllo, la stringa di fonemi contenuta nel buffer stringaFonemi può essere "letta ad alta voce" dal dispositivo Narrator. Per ottenerne la riproduzione vocale, il task deve indicare l'indirizzo del buffer stringaFonemi nella struttura di I/O e inviare il comando CMD_WRITE.

Argomenti della funzione

stringaTesto Rappresenta l'indirizzo del buffer contenente la stringa di testo da tradurre in fonemi (in C il tipo è APTR).

lunghezza_stringaTesto Indica il numero di caratteri presenti nella stringa di testo, cioè la lunghezza in byte della stringa individuata da stringaTesto. Questo parametro dev'essere di tipo LONG.

stringaFonemi Rappresenta l'indirizzo del buffer nel quale la funzione Translate inserisce le parole trasformate in fonemi. La trasformazione genera sempre una stringa più lunga di quella che contiene il testo originale. Per questo motivo, se il task non calcola con precisione la lunghezza del buffer stringaFonemi, può accadere che il buffer si riempia completamente prima che la traduzione sia completa. In questo caso Translate restituisce un offset negativo che il task deve sottrarre a stringaTesto e sommare a lunghezza_stringaTesto prima di chiamare nuovamente la funzione. Si ricordi (se una chiamata non basta) che Translate a ogni esecuzione sovrascrive il contenuto del buffer stringaFonemi, e che per conoscere la lunghezza della stringa di fonemi si può impiegare la funzione standard strlen del C.

lunghezza_stringaFonemi Rappresenta il massimo numero di caratteri che la funzione Translate può inserire nel buffer stringaFonemi. Si noti che Translate considera come limite massimo il valore indicato dal task meno 6.

Discussione

La funzione Translate è l'unica funzione disponibile nella libreria Translator. Perché il task possa chiamarla occorre che nella variabile globale TranslatorBase sia memorizzato l'indirizzo base della libreria Translator; il task ottiene questo indirizzo chiamando la funzione OpenLibrary.

La funzione Translate in pratica viene utilizzata solo per inviare una stringa di fonemi al dispositivo Narrator. Per il dispositivo Narrator il modo con cui il task crea la stringa di fonemi non ha importanza: l'unica condizione è che sia composta da fonemi organizzati secondo la particolare sintassi del dispositivo. La trattazione dei fonemi e della loro sintassi esula dagli scopi di questo volume.

Come già detto, conviene prevedere per la stringa di fonemi un buffer più capiente di quello destinato alla stringa di testo, a causa dell'aumento di dimensioni che si ottiene passando da caratteri ASCII a fonemi.

COMANDI STANDARD DEL DISPOSITIVO

CMD_FLUSH

Scopo del comando

Questo comando elimina tutte le richieste di I/O, attive o accodate che siano. Nel dispositivo Narrator questo comando agisce principalmente sui comandi CMD_WRITE e CMD_READ. CMD_FLUSH azzera sempre il parametro io_Error della struttura narrator_rb che lo definisce.

Preparazione della struttura narrator_rb

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il numero d'ordine che il task ottiene aprendo il dispositivo con la funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_FLUSH e azzerare il parametro io_Flags.

Discussione

Il comando CMD_FLUSH elimina tutte le richieste di I/O, attive o accodate alla request port del dispositivo. Dal momento che gli unici comandi che il dispositivo Narrator può accodare ed eseguire in modo asincrono sono CMD_WRITE e CMD_READ, CMD_FLUSH elimina solo questo tipo di richieste. Se si desidera eliminare solo una particolare richiesta di I/O, conviene utilizzare la funzione AbortIO.

CMD_READ

Scopo del comando

Questo comando serve per ottenere i dati relativi alla sagoma della bocca che corrisponde alla sillaba pronunciata dalla macchina. Il task, dopo aver avviato la riproduzione vocale di un testo, può inviare ripetuti comandi CMD_READ per ottenere dal dispositivo sempre nuove sagome e utilizzarle per creare l'animazione di una bocca; se l'animazione sfrutta a fondo le capacità grafiche dell'Amiga, si può ottenere un effetto molto realistico.

Si ricordi che il dispositivo genera i dati della bocca solo se il task ha inviato il relativo comando CMD_WRITE impostando a 1 il parametro mouths della struttura di I/O narrator_rb.

CMD_READ non restituisce la risposta fino a quando le routine interne del dispositivo non generano una nuova sagoma. In questo modo il task ha la sicurezza di ricevere sempre una sagoma diversa dalla precedente. Per questa ragione i task inviano i comandi CMD_READ quasi sempre in modo sincrono (funzione DoIO) per ottenere il controllo solo quando è ora di visualizzare una nuova sagoma della bocca. Questo consente di aumentare l'efficienza del sistema, ma ovviamente è possibile solo se il comando CMD_WRITE che ha dato inizio alla riproduzione vocale è stato inviato in modo asincrono (funzione SendIO).

Per inviare CMD_READ il task deve impiegare la struttura mouth_rb, versione più estesa della struttura di I/O narrator_rb. Nei parametri addizionali di mouth_rb il dispositivo restituisce i dati geometrici della nuova sagoma. Il task deve quindi allocare in memoria una struttura di tipo mouth_rb, che utilizza solo per i comandi CMD_READ.

Il primo elemento della struttura mouth_rb è una sotto-struttura di tipo narrator_rb, all'interno della quale il task deve copiare tutti i parametri che ha impiegato per inviare il comando CMD_WRITE; questo assicura che pur usando una diversa struttura di I/O, la richiesta è strutturata in modo corretto (ovviamente il task deve inserire nel parametro io_Command il comando CMD_READ).

Se non c'è nessun comando CMD_WRITE in esecuzione o accodato alla request port del dispositivo quando il task invia CMD_READ, la struttura di I/O utilizzata per inviare CMD_READ viene subito restituita al task con il codice d'errore ND_NoWrite nel parametro io_Error. In questo caso, il dispositivo non accede ai dati forniti nella struttura mouth_rb.

I risultati prodotti da questo comando sono i seguenti:

- width. Indica l'ampiezza della sagoma per la bocca espressa in millimetri/3,67. La divisione per il coefficiente 3,67 è necessaria a causa della differenza tra altezza e larghezza dei pixel. Il valore restituito dal parametro varia fra 0 e 14.

- height. Indica l'altezza della sagoma della bocca espressa in millimetri. Il valore restituito dal parametro varia fra 0 e 14.

- shape. Rappresenta la forma della bocca. Viene utilizzato soltanto dal dispositivo e non è d'interesse per il task.

- io_Error. Questo parametro contiene il codice d'errore restituito dal comando CMD_READ; il valore 0 indica che il comando è stato eseguito senza problemi; ND_NoWrite indica che CMD_READ non può essere eseguito in quanto il dispositivo Narrator non sta eseguendo nessun comando CMD_WRITE.

Preparazione della struttura mouth_rb

Il task deve copiare la struttura narrator_rb utilizzata per inviare il comando CMD_WRITE nella struttura mouth_rb che rappresenta i comandi CMD_READ; questa operazione assicura che i parametri io_Device e io_Unit contengano i valori che il task ha ottenuto aprendo il dispositivo. Se si desidera impiegare una diversa reply port per ricevere le risposte ai comandi CMD_READ, si deve memorizzarne l'indirizzo nel parametro mn_ReplyPort. Si deve infine inizializzare io_Command a CMD_READ e i parametri io_Flags, width e height a 0.

Discussione

Il comando CMD_READ permette a un task di ottenere i dati necessari per l'animazione di una bocca in concomitanza con la riproduzione vocale delle parole. La correlazione tra le sillabe pronunciate e le sagome della bocca restituite dai comandi CMD_READ è mantenuta da un apposito algoritmo interno del dispositivo Narrator.

Per ottenere questi dati, il task deve inviare per ogni comando CMD_WRITE un comando CMD_READ. In genere si possono seguire due

strade. 1) Realizzare un ciclo che termina quando la funzione CheckIO segnala la restituzione della struttura narrator_rb impiegata per inviare il comando CMD_WRITE. 2) Realizzare un loop che continua fino a quando il task non rileva il codice d'errore ND_NoWrite nel parametro io_Error della struttura mouth_rb restituita dal dispositivo; questo errore significa infatti che la riproduzione vocale è terminata e che quindi non sono disponibili sagome diverse dall'ultima.

CMD_RESET

Scopo del comando

CMD_RESET riporta l'intero dispositivo alle condizioni iniziali. Quest'operazione comporta l'esecuzione automatica del comando CMD_FLUSH e l'esecuzione automatica del comando CMD_START se il task ha inviato il comando CMD_STOP. CMD_RESET azzera sempre il parametro io_Error della struttura narrator_rb.

Preparazione della struttura narrator_rb

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il numero d'ordine che il task ottiene aprendo il dispositivo. Il task deve inizializzare io_Command con il comando CMD_RESET e azzerare il parametro io_Flags.

Discussione

CMD_RESET è un comando che ha effetti distruttivi: chiama CMD_FLUSH per eliminare la richiesta in corso di elaborazione e tutte quelle accodate. Chiama inoltre CMD_START per riattivare l'unità, nel caso che sia stata bloccata da un precedente comando CMD_STOP.

CMD_START

Scopo del comando

Se l'unità del dispositivo è stata bloccata con il comando CMD_STOP, CMD_START la riattiva. Se quando il task ha inviato il comando CMD_STOP era in corso la riproduzione vocale di un testo, ricevendo il comando CMD_START il dispositivo la riprende dal punto in cui era stata interrotta. Inoltre il comando CMD_START ordina al dispositivo Narrator di riprendere la gestione della coda alla sua request port: le richieste di I/O in essa presenti riprendono la loro ascesa verso la cima della coda. CMD_START azzera sempre il parametro io_Error della struttura narrator_rb.

Preparazione della struttura narrator_rb

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il numero d'ordine che il task ottiene aprendo il dispositivo. Il task deve inizializzare io_Command con il comando CMD_START e azzerare il parametro io_Flags.

Discussione

Il comando CMD_START fa riprendere all'unità le attività bloccate dal comando CMD_STOP. In particolare, se il comando CMD_STOP aveva sospeso una riproduzione vocale, il comando CMD_START la fa riprendere dal punto esatto in cui era stata interrotta. Si ricordi che anche il comando CMD_RESET riattiva l'unità, ma non fa riprendere la riproduzione in corso ed elimina le richieste di I/O presenti nella coda alla request port.

CMD_STOP

Scopo del comando

Il comando CMD_STOP blocca l'esecuzione di un comando CMD_WRITE nell'unità del dispositivo Narrator e quindi anche l'ascesa delle richieste di I/O presenti nella coda alla request port. Il sistema continua ad accodare tutte le richieste di I/O che giungono al dispositivo, ma l'unità non le elabora fino a quando il task non invia il comando CMD_START. CMD_STOP azzera sempre il parametro io_Error della struttura narrator_rb.

Preparazione della struttura narrator_rb

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il numero d'ordine che il task ottiene aprendo il dispositivo. Il task deve inizializzare io_Command con il comando CMD_STOP e azzerare il parametro io_Flags.

Discussione

Il comando CMD_STOP blocca alla prima occasione favorevole qualsiasi comando CMD_WRITE in esecuzione nell'unità 0. Impedisce inoltre che avvenga l'elaborazione delle richieste CMD_WRITE o CMD_READ accodate. Perché l'attività riprenda, il task deve inviare il comando CMD_START.

CMD_WRITE

Scopo del comando

Questo comando dà l'avvio alla riproduzione vocale della stringa di fonemi indicata dal task nella sua struttura di I/O. Ogni volta che arriva un comando CMD_WRITE, il dispositivo Narrator comunica con il dispositivo Audio per assegnare una delle combinazioni di canali indicate dal task; la priorità di

default è 75 e non può essere variata dal task. Se questa operazione ha successo inizia la riproduzione vocale del testo attraverso la combinazione di canali audio che Narrator è riuscito ad aprire. Quando la lettura è terminata, il dispositivo Narrator chiude l'accesso alla combinazione di canali perché altri task con più bassa priorità la possano impiegare.

Le routine interne del dispositivo Narrator accodano automaticamente le richieste CMD_WRITE che non possono essere elaborate subito.

Quando il dispositivo elabora un comando CMD_WRITE inviato dal task con il valore 1 nel parametro mouths della struttura di I/O, si predispone per soddisfare i comandi CMD_READ che riceverà dal task.

Il task in genere invia il comando CMD_WRITE in modo asincrono (cioè con le funzioni BeginIO o SendIO), così da poter eseguire altri compiti durante la riproduzione vocale del testo. Il dispositivo restituisce la struttura di I/O del comando CMD_WRITE solo quando la riproduzione termina o quando viene interrotta. Un'interruzione può verificarsi per esempio nel caso che un task richieda al dispositivo Audio l'accesso a uno dei canali audio usati da Narrator indicando una priorità superiore a 75. Allora il dispositivo Narrator restituisce il codice d'errore IOERR_ABORTED nel parametro io_Error della struttura di I/O relativa al comando CMD_WRITE. Gli altri codici d'errore che CMD_WRITE può restituire sono i seguenti:

- ND_CantAlloc. Le routine interne del dispositivo Narrator non sono riuscite ad assegnare nessuna delle combinazione di canali indicate dal task nell'array puntato da ch_masks. Questo errore viene restituito se in tutte le combinazioni che il task ha indicato al dispositivo Narrator almeno un canale risulta assegnato a un altro task con priorità superiore a 75.

- ND_PhonErr. Si è verificato un errore di "traduzione" della stringa di fonemi durante l'esecuzione di un comando CMD_WRITE. Le routine interne del dispositivo Narrator, leggendo la stringa di fonemi, hanno incontrato una sintassi che non sono in grado di riconoscere. In questo caso, il dispositivo restituisce nel parametro io_Actual il numero di lettere pronunciate fino a quel momento. Questo errore non può verificarsi se la stringa di fonemi è stata ottenuta tramite la funzione Translate.

- ND_RateErr. La velocità di riproduzione richiesta dal task non rientra nei limiti consentiti (40-400 parole al minuto).

- ND_PitchErr. La frequenza media della voce richiesta dal task non rientra nei limiti consentiti (65-320 Hz).

- ND_SexErr. Il parametro sex della struttura narrator_rb non è stato inizializzato correttamente (gli unici valori permessi sono MALE e FEMALE).

- ND_ModeErr. Il modo di riproduzione richiesto non è consentito (gli unici valori possibili sono NATURALF0 e ROBOTICF0).

- ND_FreqErr. La frequenza di campionamento indicata dal task non rientra nei limiti consentiti (5.000-28.000 dati-campione al secondo).

- ND_VolErr. Il volume indicato dal task per la voce non rientra nei limiti consentiti (0-64).

Oltre ai codici d'errore, il comando CMD_WRITE restituisce aggiornato il parametro io_Actual.

- io_Actual. Indica il numero di caratteri elaborati dal comando CMD_WRITE. Se il parametro io_Error nella struttura narrator_rb indica un errore di riconoscimento (ND_PhonErr), io_Actual permette al task di identificare il punto esatto della stringa in cui si è verificato l'errore. Il task può correggere la stringa di fonemi e inviarla nuovamente con CMD_WRITE ripartendo dal momento dell'interruzione.

Preparazione della struttura narrator_rb

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il numero d'ordine che il task ottiene aprendo il dispositivo. Il task deve inizializzare inoltre i seguenti parametri:

- io_Command. Si deve inizializzare questo parametro a CMD_WRITE.

- io_Data. Si deve inizializzare questo parametro con l'indirizzo del buffer contenente la stringa di fonemi che il dispositivo Narrator deve riprodurre. La stringa può essere ottenuta impiegando la funzione Translate o in qualsiasi altro modo. Si tratta di una stringa di caratteri ASCII che deve seguire una particolare sintassi. Se la sintassi risulta errata, il dispositivo restituisce il codice d'errore ND_PhonErr e interrompe la riproduzione.

- io_Length. Si deve inizializzare questo parametro con il numero di caratteri contenuti nella stringa di fonemi, dimensione che si può per esempio ottenere utilizzando la funzione standard strlen del C.

- rate. Si deve inizializzare questo parametro con il valore della velocità di lettura (può variare tra 40 e 400 parole al minuto).

- pitch. Si deve inizializzare questo parametro con il valore della frequenza media della voce (può variare tra 65 e 320 Hz).

- mode. Si deve inizializzare questo parametro con un numero che indica in che modo dev'essere riprodotta la voce. Può assumere il valore 0 (NATURALF0) per una voce naturale oppure 1 (ROBOTICF0) per una voce metallica.

- sex. Si deve inizializzare questo parametro con un numero che indica il sesso della voce. Può assumere il valore 0 (MALE) per una voce maschile oppure 1 (FEMALE) per una voce femminile.

- ch_masks. In questo puntatore il task deve memorizzare l'indirizzo del suo array di combinazioni dei canali. Il dispositivo Narrator impiega quest'array per accedere ai canali del dispositivo Audio quando deve eseguire un comando CMD_WRITE. Ogni byte di quest'array deve contenere nel nibble meno significativo un valore compreso fra 1 e 15, le 15 combinazioni possibili dei quattro canali audio dell'Amiga. Quando il dispositivo Narrator cerca di aprire il dispositivo Audio e di assegnarne i canali, indica tramite l'array le combinazioni di canali richieste.

- nm_masks. In questo parametro il task deve memorizzare il numero di byte contenuti nell'array. Ogni byte corrisponde a una combinazione.

- volume. Questo parametro controlla il volume della voce, e può contenere un valore compreso fra 0 (attenuazione massima del segnale) e 64 (attenuazione minima del segnale).

- sampfreq. Questo parametro indica la frequenza di campionamento espressa in dati-campione al secondo (può variare tra 5.000 e 28.000).

- mouths. Questo parametro dev'essere inizializzato a 1 se il task desidera che il dispositivo invii i dati dell'immagine di una bocca ogni volta che riceve il comando CMD_READ. I dati vengono generati solo durante la riproduzione e corrispondono alla sillaba che il dispositivo Audio sta riproducendo, cioè simulano i movimenti di una bocca reale. Se il task non desidera questi dati, deve impostare il parametro mouths a zero.

Discussione

CMD_WRITE è il comando che serve ad avviare la riproduzione vocale di una stringa di fonemi. Qualsiasi task può inviare un flusso continuo di comandi CMD_WRITE all'unità 0 del dispositivo Narrator.

I comandi CMD_WRITE vengono sempre accodati nella request port dell'unità, in quanto non è previsto il QuickIO. Quindi le stringhe di fonemi

vengono sempre lette dalla macchina nell'ordine in cui sono state accodate.

Per eliminare un comando CMD_WRITE già inviato, il task può impiegare i comandi CMD_FLUSH, CMD_RESET e la funzione AbortIO. La differenza è che i comandi CMD_FLUSH e CMD_RESET eliminano indistintamente tutte le richieste di I/O inviate dal task, mentre la funzione AbortIO permette di eliminarne una sola.

Per ciascun comando CMD_WRITE si può inviare una serie di comandi CMD_READ che di solito vengono inviati all'interno di un ciclo che continua fino a quando non sono finite le parole da riprodurre.

Il dispositivo Parallel

Introduzione

Il dispositivo Parallel permette ai task di comunicare con periferiche hardware connesse all'Amiga attraverso la porta parallela. Risiede su disco e dev'essere presente nella directory logica DEVS:. Quando infatti un task chiama OpenDevice per aprirlo, e il sistema rileva che non si trova in memoria (cioè non è ancora stato caricato oppure è stato caricato ma successivamente eliminato tramite RemDevice), procede a caricarlo da disco cercandolo nella directory logica DEVS:. Una volta che il dispositivo è stato caricato in memoria, rimane disponibile fino al reset della macchina o fino alla sua eliminazione tramite RemDevice.

Generalmente il dispositivo Parallel viene impiegato per pilotare una stampante connessa alla porta parallela dell'Amiga. È consentito il modo di accesso condiviso.

Il dispositivo Parallel permette ai task di specificare fino a otto caratteri di EOF non standard, che vengono impiegati per sospendere l'acquisizione di dati dalla periferica connessa alla porta parallela. Quando uno di questi caratteri viene rilevato nel flusso entrante, il dispositivo sospende la ricezione e restituisce al task la struttura che caratterizza la richiesta di lettura dati. Questa gestione permette a un task di comunicare con dispositivi hardware che inviano dati in pacchetti delimitati da caratteri prestabiliti (i caratteri di EOF possono assumere qualunque valore compreso fra 0x00 e 0xFF).

Il dispositivo Parallel si aspetta che i task inviino i comandi impiegando la struttura di I/O non standard IOExtPar. Alla struttura di I/O standard estesa (IOStdReq, che costituisce il primo elemento della struttura IOExtPar) IOExtPar aggiunge alcuni parametri tipici del dispositivo Parallel, come io_Status e la sotto-struttura io_PTermArray di tipo IOPArray.

Le operazioni di lettura e scrittura

Le operazioni di lettura e scrittura previste dal dispositivo Parallel sono controllate da due comandi: CMD_READ e CMD_WRITE. La Figura 5.1 (a pagina 173) illustra in che modo avviene un'operazione di lettura o di scrittura.

Il dispositivo Parallel non possiede buffer di transito interni, e comunica esclusivamente grazie ai due buffer definiti dal task, uno per le operazioni di lettura e l'altro per le operazioni di scrittura. Il parametro io_Data della struttura IOExtPar viene utilizzato dal dispositivo come puntatore all'uno o all'altro buffer, a seconda del comando.

Come viene mostrato nella Figura 5.1, il trasferimento dei dati verso l'interno e verso l'esterno di questi buffer può essere bloccato e riattivato dal task inviando i comandi CMD_STOP e CMD_START, che intervengono sulla sequenza di handshaking in corso con la periferica, in modo che anche questa venga informata della sospensione e ripresa delle comunicazioni.

A500 e A2000

Pin numero	Denominazione	Descrizione
1	DRDY	Dato pronto (output)
2	D0	Bit 0 del dato
3	D1	Bit 1 del dato
4	D2	Bit 2 del dato
5	D3	Bit 3 del dato
6	D4	Bit 4 del dato
7	D5	Bit 5 del dato
8	D6	Bit 6 del dato
9	D7	Bit 7 del dato
10	ACK	Dato ricevuto (input)
11	BUSY	Buffer della stampante pieno
12	POUT	Fine carta
13	SEL	Select – stampante on-line
14	+5 volt	Alimentazione
15	–	Non usato
16	RESET	Pin di reset dell'Amiga
17-25	GND	Massa del sistema

A1000

1	DRDY	Dato pronto (output)
2	D0	Bit 0 del dato
3	D1	Bit 1 del dato
4	D2	Bit 2 del dato
5	D3	Bit 3 del dato
6	D4	Bit 4 del dato
7	D5	Bit 5 del dato
8	D6	Bit 6 del dato
9	D7	Bit 7 del dato
10	ACK	Dato ricevuto (input)
11	BUSY	Buffer della stampante pieno
12	POUT	Fine carta
13	SEL	Select – stampante on-line
14-22	GND	Massa del sistema
23	+5 volt	Alimentazione (max 100 mA)
24	–	Non usato
25	RESET	Pin di reset dell'Amiga

Tavola 5.1: *I pin della porta parallela*

Dal momento che il dispositivo Parallel non possiede nessun buffer interno, il comando CMD_CLEAR non ha alcun effetto (inserirlo in un programma fa eseguire soltanto l'istruzione Assembly RTS). I dati vengono trasferiti

Tavola 5.2: *Bit del registro di stato della porta parallela*

Bit	Significato
0	Stampante occupata (bit = 0)
1	Carta terminata (bit = 0)
2	A500 e A2000: stampante selezionata sulla porta parallela, indicatore di chiamata sulla porta seriale
	A1000: stampante selezionata (bit = 1)
3	Lettura (bit = 0); scrittura (bit = 1)
4-7	Riservati

direttamente attraverso il registro dati della porta parallela senza sostare in un buffer intermedio.

Quando il dispositivo Parallel esegue un'operazione di scrittura (CMD_WRITE), i dati presenti nel buffer indicato dal task passano un byte alla volta attraverso il registro dati della porta parallela, e quindi all'hardware esterno a essa collegato. Il connettore della porta parallela dell'Amiga è composto da 25 pin ed è situato per tutti i modelli sul lato posteriore del cabinet. La Tavola 5.1 descrive le connessioni dei diversi pin; usando la porta parallela occorre servirsi di un cavo compatibile con questa organizzazione dei pin.

Le routine interne del dispositivo Parallel mantengono aggiornato un valore nel registro di stato della porta parallela che riporta informazioni sul flusso delle operazioni in corso e sullo stato della porta; questo valore viene restituito nel parametro io_Status della struttura IOExtPar quando il task invia il comando PDCMD_QUERY. Il significato di ogni bit è riportato nella Tavola 5.2.

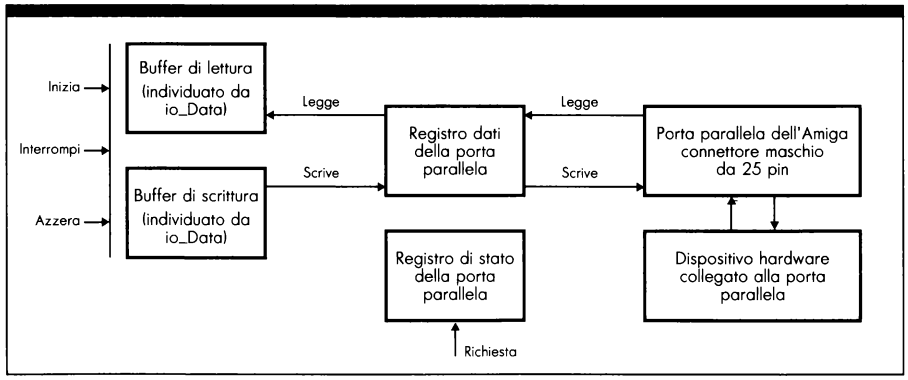


Figura 5.1: *Operazioni di lettura e scrittura del dispositivo Parallel*

comandi del dispositivo Parallel

Il dispositivo Parallel prevede nove comandi, sette standard e due specifici. L'unico comando standard non riconosciuto dal dispositivo Parallel è CMD_UPDATE, per via dell'assenza di buffer.

PDCMD_QUERY viene utilizzato per ricevere informazioni sulle condizioni del sistema e quindi svolge una funzione di controllo, senza cioè produrre alterazioni; tutti gli altri comandi, invece, effettuano cambiamenti nel dispositivo. I comandi CMD_READ e CMD_WRITE possono essere accodati oppure inviati richiedendo il QuickIO, mentre tutti gli altri vengono eseguiti in modo immediato. Ci sono soltanto due comandi che non restituiscono un codice d'errore nel parametro io_Error della struttura IOExtPar: CMD_CLEAR e PDCMD_QUERY.

L'invio dei comandi al dispositivo Parallel

Le Figure 5.2a e 5.2b (nella pagina successiva) mostrano lo schema generale utilizzato per inviare comandi al dispositivo Parallel e per chiamarne le funzioni. Le linee con le frecce rappresentano i parametri che devono essere inizializzati dall'utente (Figura 5.2a) e quelli restituiti dalle routine interne del dispositivo (Figura 5.2b).

L'interazione con il dispositivo Parallel prevede tre fasi.

1. *Preparazione della strutura IOExtPar*. In questa fase il programmatore possiede un completo controllo. Il task inizializza i parametri della struttura IOExtPar in preparazione dell'invio di un comando alla routine interne del dispositivo Parallel. Alcuni di questi parametri sono specifici

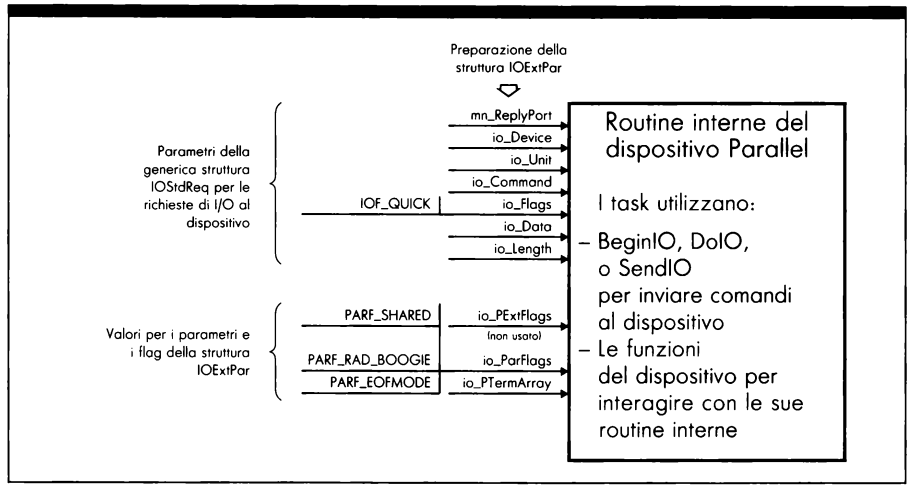


Figura 5.2a: *Gestione delle funzioni e dei comandi previsti dal dispositivo Parallel (input)*

della struttura IOExtPar, come quelli della sotto-struttura io_PTermArray, gli altri costituiscono il consueto insieme di parametri richiesti da quasi tutti i dispositivi; la scelta dei parametri dipende dal tipo di comando che s'intende inviare.

2. *Invio ed elaborazione*. L'unico compito a carico del programmatore in questa fase è inviare il comando al dispositivo utilizzando le funzioni BeginIO, DoIO o SendIO. Se la richiesta indica uno dei comandi CMD_READ o CMD_WRITE, può essere accodata oppure eseguita con il QuickIO, a seconda della richiesta (il QuickIO viene ignorato se il task ha inviato la richiesta asincronicamente, mentre viene richiesto automaticamente se il task utilizza il comando DoIO). Se invece la richiesta indica uno degli altri comandi, viene eseguita dal dispositivo sempre in modo immediato.

3. *Elaborazione dei parametri di output e loro restituzione*. Le routine interne del dispositivo Parallel aggiornano alcuni parametri della struttura di I/O e restituiscono la richiesta; questa viene accodata alla reply port del task solo se il comando era CMD_WRITE o CMD_READ e se il task non aveva richiesto il QuickIO.

Per la maggior parte dei comandi, i risultati delle elaborazioni vengono restituiti nei parametri io_Error e io_Status. Si tratta solo d'indicazioni sull'eventuale errore verificatosi e sullo stato della richiesta. I comandi CMD_WRITE e CMD_READ si servono del parametro io_Data per individuare in memoria il buffer previsto dal task.

Le Figure 5.2a e 5.2b descrivono inoltre i parametri più significativi nell'inizializzazione e nel funzionamento del dispositivo Parallel. OpenDevice e CloseDevice, per esempio, influenzano il parametro lib_OpenCnt della

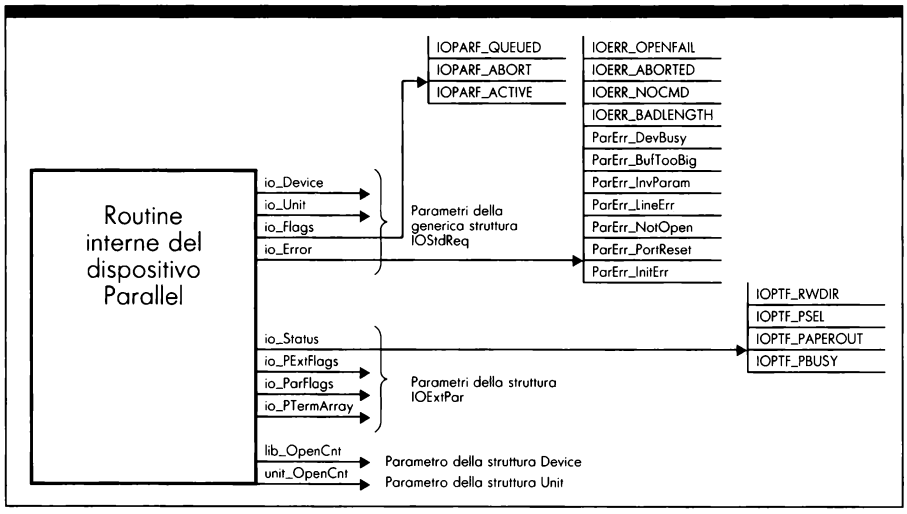


Figura 5.2b: *Gestione delle funzioni e dei comandi previsti dal dispositivo Parallel (output)*

struttura Device. A differenza del parametro io_Device, che OpenDevice aggiorna con l'indirizzo della struttura Device del dispositivo, il parametro io_Unit non viene impiegato e contiene sempre il valore 0. OpenDevice influenza anche il parametro io_Error, e può modificare il contenuto dei parametri io_ParFlags e io_PTermArray.

I parametri di default

Il dispositivo Parallel consente ai task di stabilire valori di default per due parametri della struttura IOExtPar. Il comando PDCMD_SETPARAMS, oltre a modificare il contenuto dei due parametri io_ParFlags e io_PTermArray, ordina al dispositivo di copiare al suo interno i valori indicati e di considerarli valori di default per quei parametri, cioè valori da conservare anche dopo la chiusura del dispositivo. Se in seguito un altro task apre il dispositivo, OpenDevice aggiorna con quei valori di default i parametri io_ParFlags e io_PTermArray della struttura di I/O inviata. Ovviamente, i valori di default vengono persi se il dispositivo viene eliminato dalla memoria o se riceve un altro comando-PDCMD_SETPARAMS, e possono essere variati dal task in qualunque momento.

Le strutture del dispositivo Parallel

Il dispositivo Parallel prevede due strutture: IOExtPar e IOPArray (si veda la Figura 5.3 nella pagina successiva). Si noti che la struttura IOExtPar contiene due sotto-strutture: una di tipo IOStdReq (denominata IOPar) e una di tipo IOPArray (denominata io_PTermArray). La struttura IOExtPar non contiene puntatori ad altre strutture o a zone di dati della memoria RAM. La struttura IOPArray non contiene né sotto-strutture né puntatori.

La struttura IOPArray

La struttura IOPArray è costituita da due long word nelle quali il task memorizza in sequenza i caratteri di EOF. Finora ci siamo riferiti a questo insieme di caratteri chiamandolo array ma in realtà, dal punto di vista del linguaggio C, nella struttura IOPArray non compare nessun array. D'altra parte, quando il dispositivo riceve il controllo della struttura esamina i caratteri di EOF trattando l'insieme come un array, ed è per questo che ci riferiamo a essi come se fossero contenuti in un array.

La struttura IOPArray è definita come segue:

```
struct IOPArray {
        ULONG PTermArrayØ;
        ULONG PTermArray1;
};
```

I parametri della struttura IOPArray sono i seguenti:

- PTermArray0. È costituito da 4 byte, nei quali il task definisce i primi quattro caratteri (qualsiasi numero esadecimale compreso tra 0x00 e 0xFF) che desidera indicare come caratteri di EOF. È assolutamente necessario che questi numeri vengano memorizzati in ordine decrescente.

- PTermArray1. È costituito da 4 byte, nei quali il task definisce il secondo gruppo di 4 caratteri che desidera indicare come caratteri di EOF. Di nuovo devono essere memorizzati in ordine decrescente, anche rispetto ai valori contenuti in PTermArray0. Il sistema verifica la presenza dei caratteri di EOF nel flusso entrante di dati solo se il task ha impostato il flag PARF_EOFMODE del parametro io_ParFlags.

Si noti che se vengono utilizzati meno di otto caratteri di EOF, occorre riempire la parte residua dell'array ripetendo il più basso codice ASCII utilizzato. Per esempio, l'array di codici ASCII x0807060504030303 definisce sei caratteri di EOF su otto disponibili; il task ha quindi riempito l'array ripetendo l'ultimo codice (0x03).

I caratteri di EOF impostati tramite il comando PDCMD_SETPARAMS vengono mantenuti internamente dal dispositivo anche dopo la sua chiusura da parte del task, fino a quando il dispositivo non viene rimosso dalla memoria, oppure finché non viene inviato un nuovo comando PDCMD_SETPARAMS.

La struttura IOExtPar

La struttura IOExtPar è definita come segue:

```
struct IOExtPar {
    struct IOStdReq IOPar;
    ULONG io_PExtFlags;
```

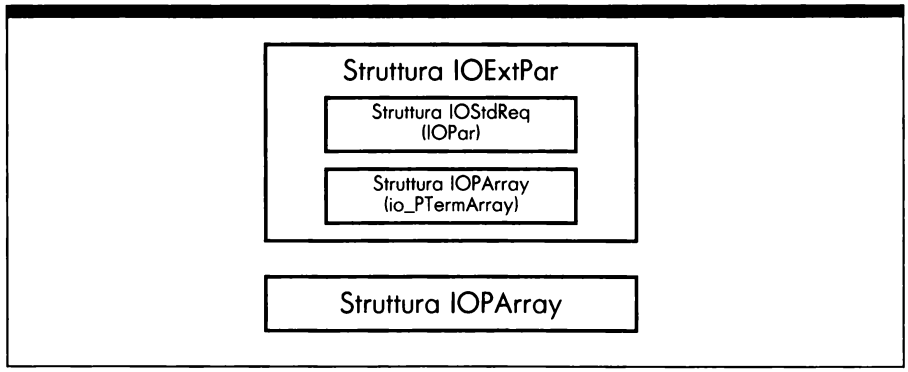


Figura 5.3: *Strutture utilizzate dal dispositivo Parallel*

```
    UBYTE io_Status;
    UBYTE io_ParFlags;
    struct IOPArray io_PTermArray;
};
```

Questi sono i parametri della struttura IOExtPar:

- IOPar. È la sotto-struttura di tipo IOStdReq che intesta il messaggio IOExtPar. Nell'intestazione assume particolare importanza il parametro mn_ReplyPort, che il task deve inizializzare con l'indirizzo della reply port (struttura MsgPort) a cui le richieste vanno accodate quando vengono restituite.

- io_PExtFlags. È un insieme di flag. Attualmente non viene utilizzato ed è presente soltanto per garantire la compatibilità con future versioni del software sistema.

- io_Status. Indica lo stato della porta parallela; viene aggiornato dal dispositivo ogni volta che perviene il comando PDCMD_QUERY.

- io_ParFlags. È costituito da un insieme di flag controllati dal task. Si veda più avanti per i loro significati. Com'è stato spiegato nel precedente paragrafo, in alcune condizioni OpenDevice inizializza questo parametro con un valore di default.

- io_PTermArray. Costituisce il nome della struttura di tipo IOPArray che permette al task di definire i propri particolari caratteri di EOF. Se questo parametro viene impostato tramite il comando PDCMD_SETPARAMS, il suo contenuto permane anche dopo la chiusura del dispositivo, fino a quando il dispositivo non viene rimosso dalla memoria, oppure finché non viene inviato un nuovo comando PDCMD_SETPARAMS.

I flag riconosciuti dal dispositivo

Il dispositivo attribuisce ai flag del parametro io_ParFlags i seguenti significati:

- PARF_SHARED (bit 5). Impostando questo flag si apre il dispositivo in modo condiviso. I task possono impostarlo prima di chiamare la funzione OpenDevice, oppure (quando il dispositivo è già stato aperto) prima d'inviare il comando PDCMD_SETPARAMS. Una volta che il dispositivo è stato aperto in modo condiviso, non è possibile azzerare il flag PARF_SHARED tramite il comando PDCMD_SETPARAMS. PARF_SHARED è l'unico flag di cui il dispositivo controlla lo stato quando il task esegue la funzione OpenDevice.

- PARF_RAD_BOOGIE (bit 3). Questo flag attiverà il trasferimento dati ad alta velocità quando sarà previsto dalle future versioni del sistema operativo. Se è stato impostato tramite il comando PDCMD_SETPARAMS, il suo stato viene mantenuto dal dispositivo anche dopo la chiusura.

- PARF_EOFMODE (bit 1). Impostando questo flag si fa in modo che il dispositivo consideri come caratteri di EOF quelli che il task indica nella sotto-struttura io_PTermArray della struttura IOExtPar. Questo flag può essere alterato dal task direttamente, senza inviare il comando PDCMD_SETPARAMS.

I flag contenuti nel parametro io_Flags hanno i seguenti significati:

- IOPARF_QUEUED (bit 6). Questo flag viene impostato quando la richiesta di I/O indica uno dei comandi CMD_READ e CMD_WRITE, ed è stata accodata.

- IOPARF_ABORT (bit 5). Questo flag viene impostato quando una richiesta CMD_READ o CMD_WRITE viene eliminata dalla funzione AbortIO oppure dal comando CMD_FLUSH.

- IOPARF_ACTIVE (bit 4). Il dispositivo mantiene questo flag impostato per tutto il tempo che la richiesta di I/O CMD_READ o CMD_WRITE risulta attiva (ossia in fase di elaborazione all'interno del dispositivo).

Il parametro io_Status della struttura IOExtPar può assumere i seguenti valori:

- IOPTF_RWDIR (bit 3). Il dispositivo restituisce questo flag impostato quando sta elaborando un comando CMD_WRITE, mentre lo restituisce azzerato quando sta elaborando un comando CMD_READ.

- IOPTF_PSEL (bit 2). Il dispositivo inizializza questo flag quando la periferica collegata alla porta parallela è pronta per ricevere dati (nel caso di una stampante quando è on-line, cioè pronta a stampare). Si tenga presente che sull'Amiga 500 e l'Amiga 2000 questo flag può risultare a 1 anche quando è attivo l'indicatore di chiamata sulla porta seriale.

- IOPTF_PAPEROUT (bit 1). Il dispositivo imposta questo flag quando la stampante segnala la mancanza di carta.

- IOPTF_PBUSY (bit 0). Il dispositivo imposta questo flag quando l'hardware collegato alla porta parallela non è in grado di ricevere dati (nel caso di una stampante, quando è off-line).

Il dispositivo Parallel restituisce nel parametro io_Error i seguenti codici d'errore:

- i codici d'errore IOERR_OPENFAIL, IOERR_ABORTED, IOERR_NOCMD e IOERR_BADLENGTH hanno il significato comune a tutti i dispositivi dell'Amiga (si veda il capitolo 3).
- ParErr_DevBusy. Il dispositivo Parallel risulta occupato. Generalmente questo errore si verifica quando il task cerca di aprire il dispositivo tramite OpenDevice ma il dispositivo risulta già aperto da un altro task in modo esclusivo.
- ParErr_BufTooBig. Si è verificato un errore nel buffer durante il trasferimento dati.
- ParErr_InvParam. È stato indicato un parametro non valido nella struttura utilizzata per inviare la richiesta.
- ParErr_LineErr. Durante il trasferimento dei dati si è verificato un errore nelle linee elettriche che collegano l'Amiga con l'hardware esterno.
- ParErr_NotOpen. Quando il comando per la richiesta di I/O è stato inviato il dispositivo Parallel non era aperto: il task deve aprirlo prima d'inviare nuovamente il comando.
- ParErr_PortReset. Il sistema è stato sottoposto a un reset.
- ParErr_InitErr. Si è verificato un errore durante l'inizializzazione del dispositivo.

Le condizioni di EOF

Il dispositivo Parallel prevede diversi sistemi per controllare la quantità di caratteri ricevuti o trasmessi lungo la linea parallela. Iniziamo descrivendo i sistemi disponibili quando si ordina al dispositivo di effettuare un'operazione di lettura, cioè di acquisire dati provenienti dalla linea parallela.

Acquisizione di dati

Il metodo più semplice per controllare l'afflusso di dati è indicare al dispositivo Parallel la quantità esatta di byte che devono essere trasferiti nel buffer di lettura del task con un comando CMD_READ. Questa quantità dev'essere memorizzata nel parametro io_Length della richiesta di I/O prima che venga inoltrata. In questo modo, il dispositivo restituisce la richiesta al task quando è pervenuto il numero di byte indicato: il dispositivo non compie

nessuna analisi dei dati in ingresso alla porta parallela, cioè nessun dato può interrompere l'afflusso. Questo metodo viene impiegato quando i dati possono essere di qualsiasi tipo, come nella ricezione di codici eseguibili.

Può invece essere necessario che il dispositivo Parallel riconosca il codice 0x00 come carattere di EOF, cioè carattere che interrompe l'afflusso dei dati. Perché questo accada, nel parametro io_Length della richiesta di I/O dev'essere memorizzato il valore −1. In questo modo, il dispositivo Parallel accetta caratteri fino a quando non rileva nel flusso il carattere 0x00. Si tratta di un metodo rischioso, in quanto il dispositivo non conosce i limiti di capienza del buffer messo a disposizione dal task e quindi non può rilevare un suo eventuale overflow. Quando il task riottiene la richiesta di I/O (il dispositivo ha rilevato uno 0x00 nel flusso entrante), può esaminare il parametro io_Actual per sapere quanti byte sono stati ricevuti.

Oltre a questi due metodi base, il task può indicare al dispositivo fino a otto caratteri di EOF "speciali", che il dispositivo prende in considerazione se il task ha impostato il flag PARF_EOFMODE nella richiesta di I/O che definisce il comando CMD_READ. In questo caso, qualunque sia il numero contenuto nel parametro io_Length, se il dispositivo rileva nel flusso entrante la presenza di uno dei caratteri di EOF speciali, sospende la ricezione e restituisce la richiesta di I/O al task.

Trasmissione di dati

Nella trasmissione il dispositivo Parallel non consente d'indicare caratteri di EOF speciali. Quindi ci sono soltanto due metodi per controllare il numero di caratteri che il dispositivo invia in seguito a un comando CMD_WRITE: indicare nel parametro io_Length il numero di byte da trasferire, oppure indicare nel parametro io_Length il valore −1 e assicurarsi che il parametro io_Data della richiesta di I/O individui un buffer in memoria nel quale sia stato inserito un opportuno byte a zero per indicare la fine della trasmissione.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOExtPar)
A1

Scopo della funzione

Questa funzione chiude l'accesso da parte del task all'unità 0. Trattandosi dell'unica unità presente nel dispositivo Parallel, l'esecuzione di questa funzione corrisponde quindi a chiudere l'accesso all'intero dispositivo. CloseDevice decrementa di 1 il parametro lib_OpenCnt della struttura Device che definisce il dispositivo. Se durante l'esecuzione di CloseDevice questo parametro arriva a 0 (nessun task sta impiegando il dispositivo) ed è presente una richiesta di eliminazione del dispositivo (ovvero è stata chiamata la funzione RemDevice all'interno dello stesso task prima di chiamare CloseDevice, oppure all'interno di un altro task), la libreria di routine del dispositivo Parallel viene completamente eliminata dalla RAM, e la struttura Device che definisce il dispositivo viene rimossa dalla lista di sistema DeviceList. Se queste operazioni vengono eseguite, alla successiva chiamata di OpenDevice per aprire il dispositivo Parallel, il sistema provvede automaticamente a caricarlo da disco.

CloseDevice aggiorna il puntatore io_Device con il valore −1, per indicare al task che se desidera riaprire il dispositivo deve eseguire nuovamente la funzione OpenDevice.

Argomenti della funzione

iOExtPar — In questo argomento il task deve indicare la struttura di tipo IOExtPar che ha impiegato per aprire il dispositivo con la funzione OpenDevice.

Discussione

CloseDevice chiude l'accesso al dispositivo Parallel e al dispositivo Timer, che viene aperto indirettamente durante l'esecuzione della funzione OpenDevice. Se il dispositivo viene aperto in modo esclusivo, nessun task può accedervi finché non viene chiamata CloseDevice.

Prima di chiamare CloseDevice si deve sempre verificare che tutte le richieste di I/O siano state restituite dalle routine interne del dispositivo. Per effettuare questa operazione, il task deve impiegare le funzioni GetMsg, Remove, CheckIO e WaitIO.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("parallel.device", ØL, iOExtPar, ØL)
DØ                 AØ                DØ A1       D1
```

Scopo della funzione

Questa funzione apre l'accesso all'unità 0 del dispositivo Parallel e provvede anche ad aprire l'accesso al dispositivo Timer per temporizzare opportunamente la gestione della porta parallela. Prima di chiamare la funzione, il task può impostare il flag PARF_SHARED nel parametro io_ParFlags della struttura di I/O, in modo che il dispositivo venga aperto in modo condiviso. Se invece tale flag è a zero, OpenDevice apre il dispositivo in modo esclusivo.

Nel caso che non riesca ad aprire il dispositivo, la funzione restituisce un codice d'errore diverso da zero: viene restituito il codice d'errore IOERR_OPENFAIL se il dispositivo non si trova in memoria e non risulta nemmeno nella directory logica DEVS:, e il codice d'errore ParErr_DevBusy se il dispositivo risulta aperto da un altro task in modo esclusivo.

Dopo l'apertura del dispositivo, OpenDevice inizializza alcuni parametri della struttura IOExtPar. Oltre ai consueti io_Device e io_Unit (quest'ultimo viene semplicemente azzerato), OpenDevice aggiorna i parametri io_ParFlags e io_PTermArray con i più recenti valori di default che il dispositivo ha memorizzato (si tratta dei valori di default impostati dall'ultima esecuzione del comando PDCMD_SETPARAMS). Se invece il dispositivo non ha mai eseguito il comando PDCMD_SETPARAMS da quando è stato caricato in memoria, tali parametri non vengono modificati. Infine, OpenDevice provvede a incrementare il parametro lib_OpenCnt contenuto nella struttura Device che gestisce il dispositivo.

Prima di chiamare OpenDevice, il task, oltre a impostare eventualmente il flag PARF_SHARED del parametro io_ParFlags, può già indicare nel parametro mn_ReplyPort l'indirizzo della sua reply port. Ai fini dell'apertura del dispositivo questa operazione non è comunque indispensabile. Il task può infatti aggiornare quel parametro dopo l'apertura o perfino decidere di non indicare nessuna reply port nelle richieste di I/O che invia al dispositivo (in questo caso il dispositivo non restituisce nulla). Per allocare una message port da impiegare come reply port, il task può utilizzare la funzione CreatePort (di supporto alla libreria Exec) la quale alloca anche un bit di segnale nella struttura Task del task e memorizza il suo numero nella message port, in modo

che il task venga avvertito quando giunge un messaggio.
I parametri restituiti dalla funzione sono i seguenti:

- io_Device. Questo puntatore viene inizializzato con l'indirizzo della struttura Device relativa al dispositivo Parallel. Il task deve indicare questo indirizzo in ogni struttura di I/O che utilizza per comunicare con il dispositivo. La struttura Device definisce una libreria di routine e il suo indirizzo costituisce quindi l'indirizzo base della libreria, la quale rimane in memoria finché il dispositivo non viene eliminato.

- io_Unit. Questo puntatore viene semplicemente azzerato. Contrariamente a quanto esposto nei primi due capitoli, nel caso del dispositivo Parallel non viene associata all'unità 0 (l'unica che il dispositivo possieda) alcuna struttura Unit.

- io_Error. Un valore di questo parametro pari a 0 indica che l'apertura ha avuto successo. IOERR_OPENFAIL indica che non è stato possibile aprire il dispositivo Parallel (per esempio perché non si trovava nella directory logica DEVS:). Se si tenta di aprire il dispositivo nel modo di accesso esclusivo quando è già stato aperto da altri task, oppure se si tenta di aprirlo in modo condiviso quando un task lo detiene in modo esclusivo, viene restituito il codice d'errore ParErr_DevBusy. Non vengono restituiti codici d'errore se il task indica negli argomenti della chiamata un numero di unità diverso da zero: il dispositivo procede sempre ad aprire l'unità 0.

- io_ParFlags. OpenDevice accede in scrittura a questo parametro solo se è stato impartito un comando PDCMD_SETPARAMS dopo il caricamento in memoria del dispositivo, e vi memorizza il più recente valore di default impostato (senza alterare lo stato del flag PARF_SHARED deciso dal task prima di chiamare OpenDevice).

- IOPArray. OpenDevice accede in scrittura a questo parametro solo se è stato impartito un comando PDCMD_SETPARAMS dopo il caricamento in memoria del dispositivo, e vi memorizza i più recenti caratteri di EOF non standard che sono stati impostati.

Argomenti della funzione

"parallel.device" Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Parallel.

ØL Questo argomento può indicare qualunque valore. Per ragioni di compatibilità con future versioni del dispositivo, è comunque opportuno indicare il valore 0. Se infatti il dispositivo verrà modificato per agire su

più porte parallele, questo argomento si dovrà ovviamente prendere in considerazione.

iOExtPar — Questo argomento dev'essere l'indirizzo della struttura IOExtPar che il task impiega per interagire con il dispositivo.

ØL — Questo valore indica che l'argomento flag non viene preso in considerazione della funzione.

Preparazione della struttura IOExtPar

Prima di aprire il dispositivo Parallel, il task può inizializzare il parametro mn_ReplyPort con l'indirizzo della struttura MsgPort che rappresenta la sua reply port. Questa operazione non è però obbligatoria e può anche essere effettuata dopo la chiamata a OpenDevice, magari indicando una diversa reply port per ogni tipo di comando. L'altro parametro che il task può impostare è il flag PARF_SHARED se desidera aprire il dispositivo Parallel nel modo di accesso condiviso.

Discussione

La funzione OpenDevice viene utilizzata per accedere alle routine interne del dispositivo. Una volta in possesso del dispositivo Parallel, il task può impartire una serie di comandi CMD_WRITE e CMD_READ (tramite BeginIO, DoIO o SendIO) per inviare e ricevere informazioni da un dispositivo hardware collegato alla porta parallela dell'Amiga. Quando il task non ha più bisogno di accedere alla porta parallela, è opportuno che chiuda il dispositivo tramite la funzione CloseDevice, specialmente se l'aveva aperto nel modo di accesso esclusivo.

Il dispositivo Parallel può essere aperto in modo esclusivo oppure condiviso. Se un task apre il dispositivo in modo condiviso, un secondo task può a sua volta aprirlo senza attendere che il primo lo chiuda.

Si noti infine che esiste una stretta relazione tra OpenDevice e il comando PDCMD_SETPARAMS: i valori impostati da questo comando vengono infatti considerati valori di default per tutto il tempo che il dispositivo rimane in memoria (oppure finché non viene inoltrato un nuovo comando PDCMD_SETPARAMS) e OpenDevice li utilizza per aggiornare il parametro io_ParFlags e i parametri della sotto-struttura io_PTermArray.

COMANDI STANDARD DEL DISPOSITIVO

CMD_FLUSH

Scopo del comando

CMD_FLUSH elimina tutte le richieste di I/O CMD_READ e CMD_WRITE, sia accodate che in esecuzione, e viene sempre eseguito in modo immediato. Le richieste di I/O che risultano accodate vengono restituite alla reply port del task con il codice d'errore IOERR_ABORTED nel parametro io_Error.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. ParErr_InvParam indica che il task ha utilizzato un parametro non corretto nella struttura IOExtPar impiegata per inviare il comando. ParErr_NotOpen indica che il dispositivo Parallel non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'inviare nuovamente il comando CMD_FLUSH.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_FLUSH e azzerare il parametro io_Flags.

Discussione

Il comando CMD_FLUSH elimina tutte le richieste di I/O CMD_READ e CMD_WRITE, in esecuzione o accodate alla request port. Dato che CMD_FLUSH ha effetti distruttivi, si deve utilizzarlo soltanto se si desidera riportare il dispositivo alla configurazione iniziale.

CMD_READ

Scopo del comando

Il comando CMD_READ serve per ricevere dalla porta parallela una stringa di dati e memorizzarla all'interno di un buffer in memoria. Ogni volta che la periferica connessa alla porta parallela invia un byte, CMD_READ lo legge dal registro di I/O della porta e lo trasferisce nel buffer definito dal task.

Se nel parametro io_Length viene indicato −1, il dispositivo Parallel continua a leggere caratteri fino a quando non viene rilevata una condizione di EOF, che può essere un byte a zero oppure uno dei caratteri di EOF specificati dal task nella struttura IOPArray e attivati impostando il flag PARF_EOFMODE del parametro io_ParFlags.

Se nel parametro io_Length viene invece inserito un numero positivo, il dispositivo Parallel continua a leggere caratteri fino a quando non ha raggiunto la quantità indicata, anche se nel flusso entrante transitano degli zeri. In questo caso l'acquisizione dei dati può essere sospesa prima che la quantità indicata venga raggiunta solo se il task ha impostato il flag PARF_EOFMODE e nel flusso viene intercettato uno dei caratteri di EOF che il task ha indicato nella struttura IOPArray.

CMD_READ può essere trattato tanto come una richiesta di I/O sincrono (DoIO) quanto come una richiesta asincrona (SendIO).

I risultati prodotti dall'esecuzione del comando vengono restituiti nei parametri io_Actual e io_Error.

- io_Actual. Questo parametro indica il numero di caratteri effettivamente trasferiti nel buffer del task; può differire dal numero che il task ha indicato in io_Length qualora venga riscontrato nel flusso di byte uno dei caratteri di EOF indicati dalla struttura IOPArray o se il task ha indicato −1; nel primo caso il numero restituito in io_Actual conteggia anche il carattere di EOF rilevato, mentre nel secondo caso non conteggia il byte a zero che ha causato l'interruzione.

- io_Error. Un valore di questo parametro pari a 0 indica che il comando è stato eseguito. ParErr_DevBusy indica che il dispositivo Parallel era occupato e non ha potuto eseguire il comando. ParErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtPar che definiva il comando. ParErr_BufTooBig indica che il buffer di lettura definito da io_Length è di lunghezza eccessiva. ParErr_LineErr indica che ci sono dei problemi con le linee elettriche (di solito si tratta di un collegamento non corretto tra la porta parallela dell'Amiga e il dispositivo hardware esterno). ParErr_NotOpen indica che il dispositivo Parallel non era aperto e quindi il task deve eseguire OpenDevice prima d'inviare nuovamente il comando.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_READ e inizializzare i seguenti parametri.

- io_Flags. Inizializzando questo parametro a IOF_QUICK si richiede il QuickIO. In tutti gli altri casi si deve inizializzarlo a 0. Si ricordi che se si chiama la funzione DoIO il QuickIO viene richiesto automaticamente.

- io_ParFlags. Inizializzando questo parametro a PARF_EOFMODE si richiede che il comando CMD_READ riconosca gli otto caratteri di EOF che il task ha indicato nella sotto-struttura IOPArray.

- io_Length. Si deve inizializzare questo parametro con il numero dei caratteri da traferire al buffer del task. Se il task indica il valore −1, il comando legge caratteri fino a quando non rileva uno zero. Conviene sempre allocare un buffer più capiente del necessario per evitare l'overflow (il trasferimento dei dati provenienti dalla porta parallela continuerebbe sulla RAM che segue il buffer, compromettendo con ogni probabilità la funzionalità del sistema).

- io_Data. Si deve inizializzare questo parametro con l'indirizzo del buffer nel quale il task desidera siano memorizzati i byte provenienti dalla porta parallela dell'Amiga.

Discussione

CMD_READ permette a un task di ricevere in uno dei suoi buffer i dati provenienti dalla periferica collegata alla porta parallela dell'Amiga. I dati vengono trasferiti un byte alla volta dall'hardware esterno nel registro dati della porta parallela e poi nel buffer allocato dal task. L'acquisizione viene interrotta dall'arrivo di un byte a zero se nel parametro io_Length è stato memorizzato il valore −1; altrimenti, i caratteri a zero non vengono considerati.

Se è stato impostato il flag PARF_EOFMODE del parametro io_ParFlags, il trasferimento dei dati viene interrotto quando il dispositivo rileva uno dei caratteri di EOF definiti nella struttura IOPArray. Questo sistema permette al task di configurare le proprie operazioni di lettura nel modo più adatto alla periferica collegata. Per esempio, se la periferica utilizza il carattere Ctrl-Z per separare i blocchi di dati che invia, il task può definire un array di caratteri EOF composto da otto caratteri Ctrl-Z, di modo che il comando CMD_READ consideri concluso il trasferimento al termine di ogni blocco di dati.

CMD_RESET

Scopo del comando

CMD_RESET riporta tutti i parametri del dispositivo Parallel ai loro valori di default. Tutte le richieste di I/O CMD_READ e CMD_WRITE, in elaborazione o in coda, vengono eliminate e l'unità viene riattivata se era stata precedentemente bloccata con il comando CMD_STOP.

CMD_RESET viene sempre eseguito in modo immediato. Tutte le richieste di I/O eliminate vengono restituite alla reply port del task con il flag IOERR_ABORTED del parametro io_Error impostato. L'esito del comando viene restituito nel parametro io_Error. Un valore pari a 0 indica che il comando è stato eseguito. ParErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtPar utilizzata per inviare il comando. ParErr_NotOpen indica che il dispositivo Parallel non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando CMD_RESET.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_RESET e azzerare il parametro io_Flags.

Discussione

CMD_RESET è un comando che ha effetti distruttivi. Esso chiama CMD_FLUSH per eliminare tutte le richieste di I/O CMD_READ e CMD_WRITE accodate alla request port. Inoltre, CMD_RESET chiama il comando CMD_START per riattivare l'unità 0 qualora sia stata precedentemente bloccata con il comando CMD_STOP (il task non ha più bisogno d'inviare il comando CMD_FREE). CMD_RESET inoltre inizializza i parametri del dispositivo Parallel ai loro valori di default.

CMD_START

Scopo del comando

CMD_START riabilita le operazioni di scrittura e lettura bloccate con il comando CMD_STOP. Questa operazione viene compiuta riattivando il meccanismo di trasmissione della porta parallela. La riabilitazione riguarda qualsiasi comando CMD_WRITE o CMD_READ bloccato durante la propria attività. Oltre alle elaborazioni in corso, riprende anche la gestione della coda alla request port.

CMD_START viene sempre eseguito in modo immediato. L'esito del comando è riportato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. Il codice ParErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtPar utilizzata per inviare il comando. ParErr_NotOpen indica che il dispositivo Parallel non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'inviare nuovamente il comando CMD_START.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_START e azzerare il parametro io_Flags.

Discussione

CMD_START riattiva le operazioni di lettura e scrittura di dati nel registro della porta parallela, e riattiva anche la gestione della coda alla request port del dispositivo.

CMD_STOP

Scopo del comando

Il comando CMD_STOP blocca l'elaborazione di un comando CMD_WRITE o CMD_READ in esecuzione e quindi anche l'ascesa delle richieste di I/O nella coda alla request port. La sospensione dell'elaborazione causa inoltre la sospensione della sequenza di handshaking in corso con la periferica. Il congelamento della coda alla request port non pregiudica l'accodamento di altre richieste da parte del sistema, ma l'unità non le elabora fino a quando il task non invia un comando CMD_START. Si ricordi che se l'unità viene riattivata tramite il comando CMD_RESET, le richieste presenti nella coda vengono rimosse e restituite ai mittenti con il codice d'errore IOERR_ABORTED.

CMD_STOP viene sempre eseguito in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. ParErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtPar utilizzata per inviare il comando. ParErr_NotOpen indica che il dispositivo Parallel non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando CMD_STOP.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_STOP e azzerare il parametro io_Flags.

Discussione

Il comando CMD_STOP blocca l'elaborazione di un comando CMD_READ o CMD_WRITE all'interno dell'unità.

CMD_WRITE

Scopo del comando

CMD_WRITE provoca il trasferimento di una stringa di caratteri da un buffer definito dal task al registro dati del dispositivo Parallel, e quindi verso la periferica collegata alla porta parallela. Il task deve indicare il numero di caratteri da trasmettere nel parametro io_Length della struttura IOExtPar; se viene indicato −1, il dispositivo Parallel continua a trasmettere caratteri fino a quando non rileva nel buffer del task il carattere di EOF 0x00. L'esecuzione di un comando CMD_WRITE può terminare anche per altre cause (per esempio, se si verifica un errore di trasferimento). In ogni caso, il numero di byte trasferiti viene sempre restituito nel parametro io_Actual della struttura IOExtPar.

CMD_WRITE può essere trattato come una richiesta di I/O sincrono o asincrono. I risultati prodotti dall'esecuzione di questo comando sono i seguenti:

- io_Actual. Questo parametro indica il numero di caratteri effettivamente trasferiti, che può differire da quello che il task ha indicato nel parametro io_Length solo se si verifica un errore di trasmissione oppure una condizione di EOF.

- io_Error. Un valore di questo parametro pari a 0 indica che il comando è stato eseguito. ParErr_DevBusy indica che il dispositivo Parallel era occupato e non ha potuto eseguire il comando CMD_WRITE come richiesto. ParErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtPar utilizzata per inviare il comando CMD_WRITE. ParErr_BufTooBig indica che il buffer di scrittura definito da io_Length è di lunghezza eccessiva. ParErr_LineErr indica che ci sono dei problemi con le linee elettriche (di solito si tratta di un collegamento non corretto tra la porta parallela dell'Amiga e il dispositivo hardware esterno). ParErr_NotOpen indica che il dispositivo Parallel non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando CMD_WRITE.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice.

Si devono inoltre inizializzare i seguenti parametri:

- io_Command. Si deve inizializzare questo parametro a CMD_WRITE.
- io_Flags. Si deve inizializzare questo parametro a IOF_QUICK per richiedere il QuickIO; altrimenti inizializzarlo a 0. Si ricordi che se si chiama la funzione DoIO il QuickIO viene richiesto automaticamente.
- io_Length. Si deve inizializzare questo parametro con il numero dei caratteri da inviare alla porta parallela dell'Amiga, oppure inizializzarlo a −1 per indicare al task di continuare a inviare caratteri fino a quando non viene trasferito il carattere di EOF 0x00.
- io_Data. Si deve inizializzare questo parametro in modo che punti al buffer del task contenente i caratteri che devono essere inviati alla periferica attraverso il registro dati della porta parallela.

Discussione

CMD_WRITE permette a un task d'inviare dati da un buffer definito al suo interno fino al registro dati della porta parallela e quindi alla periferica. Se il task ha indicato nel parametro io_Length il numero di caratteri da trasferire, il comando invia la quantità di byte indicata senza compiere nessuna analisi sui singoli byte. Se invece il task ha indicato il valore −1 nel parametro io_Length, il dispositivo continua a trasferire caratteri fino a quando non riscontra nel buffer del task il codice 0x00, che non viene trasmesso.

COMANDI SPECIFICI DEL DISPOSITIVO

PDCMD_QUERY

Scopo del comando

Il comando PDCMD_QUERY permette a un task di rilevare lo stato della porta parallela. Per esempio, il task può inviare PDCMD_QUERY per rilevare se le routine stanno eseguendo un'operazione di lettura o scrittura di dati attraverso la porta parallela, oppure, nel caso di una stampante, se è finita la carta. Il comando fornisce le informazioni impostando gli opportuni bit del parametro io_Status della struttura IOExtPar che restituisce al task.

PDCMD_QUERY viene sempre eseguito in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. Il codice ParErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtPar utilizzata per inviare il comando. ParErr_NotOpen indica che il dispositivo Parallel non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando PDCMD_QUERY.

I significati dei flag del parametro io_Status restituiti dal dispositivo sono i seguenti:

- IOPTF_RWDIR (bit 3). Il dispositivo imposta questo flag quando sta elaborando un comando CMD_WRITE, e lo azzera quando sta elaborando un comando CMD_READ.

- IOPTF_PSEL (bit 2). Il dispositivo imposta questo flag quando la periferica collegata alla porta parallela è pronta per ricevere dati (nel caso di una stampante quando è on-line, cioè pronta a stampare). Si tenga presente che nell'Amiga 500 e nell'Amiga 2000 questo bit può risultare a 1 anche quando è attivo l'indicatore di chiamata della porta seriale.

- IOPTF_PAPEROUT (bit 1). Il dispositivo imposta questo flag quando la stampante collegata segnala la mancanza di carta.

- IOPTF_PBUSY (bit 0). Il dispositivo inizializza questo flag quando il dispositivo hardware collegato alla porta parallela non è in grado di ricevere dati (nel caso di una stampante, quando è off-line).

I bit 4-7 sono riservati per future versioni del dispositivo Parallel.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando PDCMD_QUERY e azzerare il parametro io_Flags.

Discussione

Il comando PDCMD_QUERY permette a un task di controllare l'attività del dispositivo e dell'hardware esterno collegato alla porta parallela. Di solito viene utilizzato per controllare una stampante connessa alla porta parallela e pilotata

dai comandi CMD_WRITE inviati dal task. Grazie a PDCMD_QUERY il task ottiene nel parametro io_Status diverse informazioni sulle operazioni in corso e sulle condizioni della stampante.

PDCMD_SETPARAMS

Scopo del comando

PDCMD_SETPARAMS permette a un task di aggiornare due parametri interni del dispositivo con i valori indicati nei parametri io_ParFlags e io_PTermArray. Tramite il comando PDCMD_SETPARAMS, i valori indicati dal task diventano di default, cioè diventano quelli che OpenDevice memorizza nei parametri io_ParFlags e io_PTermArray delle strutture di I/O che riceve (se da quando è stato caricato in memoria il dispositivo non ha mai eseguito il comando PDCMD_SETPARAMS, OpenDevice non memorizza in quei due parametri nessun valore).

Prima d'inviare questo comando, il task deve assicurarsi che non ci siano comandi CMD_WRITE o CMD_READ attivi o accodati, dal momento che in questi casi il comando non ha successo.

PDCMD_SETPARAMS viene eseguito sempre in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. ParErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtPar utilizzata per inviare il comando. ParErr_NotOpen indica che il dispositivo Parallel non è stato aperto, e il task deve quindi eseguire OpenDevice prima d'inviare nuovamente il comando PDCMD_SETPARAMS.

Preparazione della struttura IOExtPar

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando PDCMD_SETPARAMS, azzerare il parametro io_Flags e inizializzare i seguenti parametri:

- io_PExtFlags. Non viene utilizzato dalla versione 1.3 del software sistema e dev'essere inizializzato a 0.

- io_ParFlags. Si deve inizializzare questo parametro a PARF_SHARED

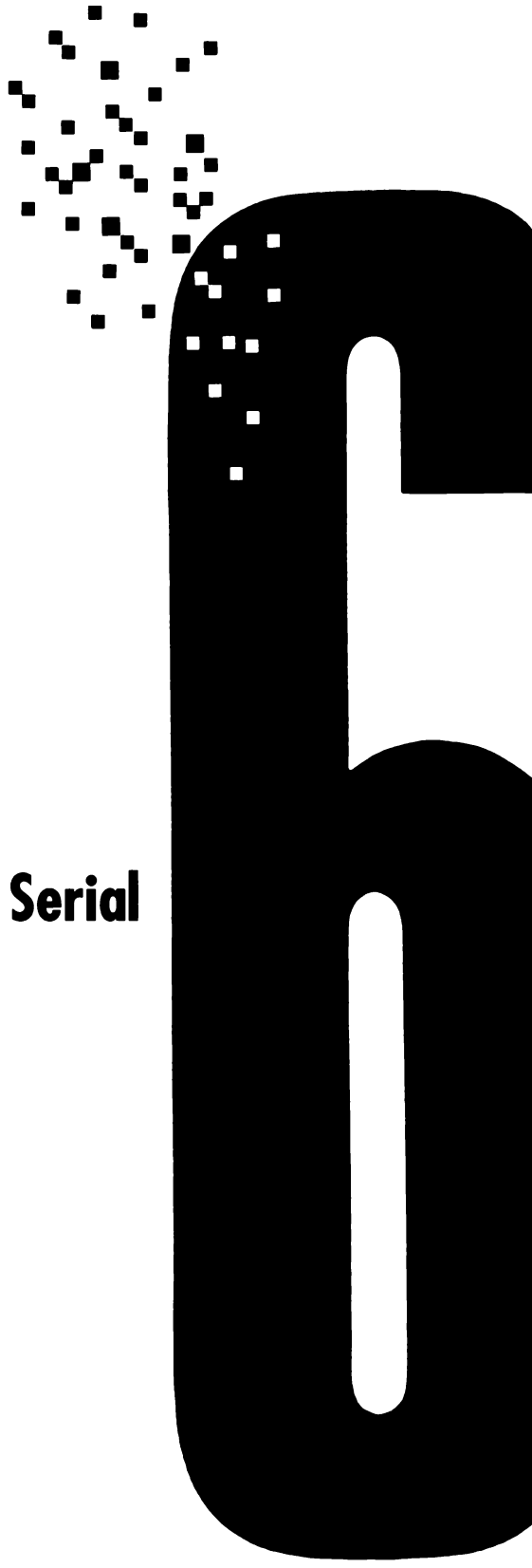
per aprire il dispositivo Parallel in modo condiviso. Impostando il flag PARF_EOFMODE si abilitano i caratteri di EOF definiti dal task nella sotto-struttura io_PTermArray. Il flag PARF_RAD_BOOGIE permetterà in future versioni di accedere al dispositivo Parallel ad alta velocità (per ora il dispositivo si limita ad azzerarlo).

- io_PTermArray. Nelle due long word di questa sotto-struttura si devono memorizzare i propri caratteri di EOF "speciali" (si ricordi che questi caratteri devono essere disposti in ordine decrescente). Questo parametro interviene nel trasferimento di dati soltanto se risulta impostato il flag PARF_EOFMODE del parametro io_ParFlags. Una volta abilitati, i caratteri di EOF restano validi anche dopo la chiusura del dispositivo.

Discussione

Il comando PDCMD_SETPARAMS permette a un task di cambiare alcuni parametri del dispositivo Parallel e riveste una particolare importanza nella definizione dei caratteri di EOF speciali per il comando CMD_READ. Esso permette a un task di alterare la definizione dei caratteri di EOF in modo da bloccare l'esecuzione di un comando CMD_READ sulla base di nuovi criteri. Se il task è in comunicazione con una serie di dispositivi hardware che richiedono blocchi di dati separati da particolari caratteri, il task può cambiare l'array dei caratteri di EOF per il successivo comando CMD_READ in modo da essere pronto per comunicare con un nuovo dispositivo hardware.

È anche possibile definire un task separato per ciascun dispositivo collagabile alla porta parallela dell'Amiga. In questo caso ogni task definirebbe i propri array di fine-trasmissione.



Il dispositivo Serial

Introduzione

Il dispositivo Serial permette ai task di comunicare con le periferiche collegate all'Amiga tramite l'interfaccia seriale. Normalmente si tratta di un altro computer collegato direttamente o tramite modem, o di una stampante seriale.

Il dispositivo Serial risiede su disco e dev'essere presente nella directory logica DEVS:. Quando un task chiama OpenDevice per aprire il dispositivo Serial e questo non si trova in memoria, il sistema procede a caricarlo da disco. Una volta caricato, il dispositivo rimane disponibile per i task fino al reset della macchina o alla sua eliminazione tramite RemDevice.

Il dispositivo Serial è per alcuni aspetti simile al dispositivo Parallel; per esempio, entrambi consentono ai task di definire un insieme di caratteri di EOF che bloccano un'operazione di lettura (ricezione di byte da un secondo computer). Tuttavia, esistono alcune importanti differenze. A parte l'ovvia diversità fra i due protocolli e sistemi di comunicazione, il dispositivo Serial possiede per esempio un buffer interno per l'esecuzione del comando CMD_READ. Si tratta del *buffer interno di lettura* del dispositivo Serial.

I comandi del dispositivo Serial permettono a qualsiasi task di aprirlo e di condividerne le routine interne, purché al momento dell'apertura venga indicato il modo d'accesso condiviso.

Il dispositivo Serial permette ai task di specificare fino a otto caratteri di EOF. Questi particolari caratteri vengono impiegati per sospendere l'acquisizione di dati provenienti da una qualsiasi periferica connessa alla porta seriale. Quando uno di questi caratteri viene rilevato nel flusso entrante, il dispositivo sospende la ricezione e restituisce al task la richiesta di lettura dati che gli era stata inviata. Questa gestione permette a un task di comunicare con dispositivi hardware che inviano pacchetti di dati delimitati da caratteri particolari (i caratteri di EOF possono assumere qualunque valore compreso fra 0x00 e 0xFF).

Le operazioni di lettura e scrittura

Le operazioni di lettura e scrittura del dispositivo Serial sono controllate da due comandi: CMD_READ e CMD_WRITE. La Figura 6.1 (nella pagina successiva) mostra in che modo avvengono queste operazioni.

Il dispositivo richiede al task di allocare in memoria due buffer, uno per le operazioni di lettura e uno per quelle di scrittura; di ciascuno il task deve indicare l'indirizzo nel puntatore io_Data della struttura IOExtSer quando invia il relativo comando (eventualmente può trattarsi dello stesso buffer, ma solo nelle comunicazioni half-duplex). Oltre a questi due buffer, il comando CMD_READ utilizza anche il già citato buffer interno del dispositivo, che generalmente occupa 512 byte (il comando CMD_WRITE non utilizza invece

nessun buffer del dispositivo per il transito dei dati).

I dati contenuti nel buffer interno di lettura possono essere cancellati utilizzando il comando CMD_RESET (disalloca il vecchio buffer e ne alloca uno nuovo con la dimensione di default) o CMD_CLEAR (azzera inoltre il puntatore al buffer); CMD_STOP e CMD_START, sospendendo e riattivando le comunicazioni, possono invece essere utilizzati per accedere ai dati che si trovano nel buffer di scrittura del task e che sono già in fase di elaborazione da parte del dispositivo.

Quando il dispositivo Serial esegue un comando CMD_READ per ricevere un flusso di dati da una periferica, tutti i byte che essa invia passano attraverso il registro dati; ogni volta che nel registro risulta presente un nuovo byte, il dispositivo lo legge e lo memorizza nel suo buffer interno.

Quando il dispositivo Serial esegue un comando CMD_WRITE per inviare un flusso di dati a una periferica, preleva un byte alla volta dal buffer di scrittura del task e lo memorizza nel registro dati in modo che venga trasmesso attraverso la porta seriale collocata sul retro dell'Amiga.

Il connettore maschio (femmina sull'Amiga 1000) della porta seriale è composto di 25 pin ed è situato sul pannello posteriore del cabinet in posizione centrale per tutti e tre i modelli. La Tavola 6.1 (nella pagina successiva) descrive le connessioni dei pin. Per utilizzare la porta seriale occorre disporre di un cavo compatibile con questa organizzazione dei pin.

Le routine interne del dispositivo Serial provvedono a mantenere aggiornato il registro di stato della porta seriale; quando il task invia il comando SDCMD_QUERY, il valore contenuto nel registro di stato viene copiato nel parametro io_Status della struttura IOExtSer relativa al comando. Il significato dei bit contenuti in questo parametro è riassunto nella Tavola 6.2 (a pagina 202).

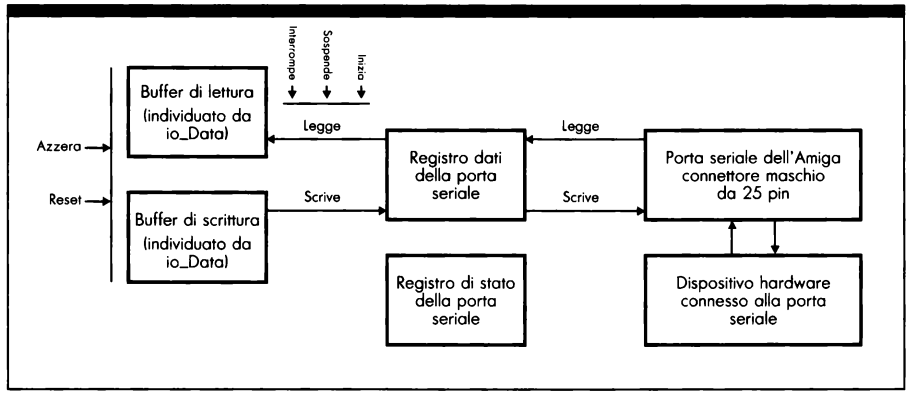


Figura 6.1: *Operazioni di lettura e scrittura del dispositivo Serial*

A500 e A2000

Pin numero	Denominazione	Descrizione
1	FGND	Massa del telaio
2	TXD	Linea dati in trasmissione
3	RXD	Linea dati in ricezione
4	RTS	Richiesta per trasmettere
5	CTS	OK per trasmettere
6	DSR	Periferica pronta
7	GND	Massa di sistema
8	CD	Portante presente
9	+12 volt	Alimentazione
10	−12 volt	Alimentazione
11	AUDO	Output audio
12-17	–	Non usati
18	AUDI	Input audio
19	–	Non usato
20	DTR	Computer pronto
21	–	Non usato
22	RI	Indicatore di chiamata
23-25	–	Non usati

A1000

1	FGND	Massa del telaio
2	TXD	Linea dati in trasmissione
3	RXD	Linea dati in ricezione
4	RTS	Richiesta per trasmettere
5	CTS	OK per trasmettere
6	DSR	Periferica pronta
7	GND	Massa di sistema
8	CD	Portante presente
9-13	–	Non usati
14	−5 volt	Alimentazione (max 50 mA)
15	AUDO	Output audio
16	AUDI	Input audio
17	EB	Clock bufferizzato (716 KHz)
18	INT2	Linea di interrupt: livello 2
19	–	Non usato
20	DTR	Computer pronto
21	+5 volt	Alimentazione (max 100 mA)
22	–	Non usato
23	+12 volt	Alimentazione (max 50 mA)
24	C2	Clock (3,58 MHz)
25	RESB	Reset di sistema

Tavola 6.1: *I pin della porta seriale*

comandi del dispositivo Serial

Per il dispositivo Serial sono previsti dieci comandi, sette dei quali sono quelli standard. SDCMD_QUERY rileva lo stato del dispositivo senza introdurre modifiche, mentre tutti gli altri comandi svolgono qualche azione che varia lo stato del dispositivo.

L'invio dei comandi al dispositivo Serial

Le Figure 6.2a e 6.2b (nella pagina successiva) mostrano lo schema generale utilizzato per inviare comandi al dispositivo Serial e per chiamarne le funzioni. Le linee con le frecce rappresentano nella prima figura i parametri da inizializzare, mentre nella seconda mostrano quelli restituiti dalle routine interne del dispositivo.

Per rendere operativo il dispositivo Serial sono previste tre fasi, a parte ovviamente l'apertura.

1. *Preparazione della struttura IOExtSer*. In questa fase si inizializzano i parametri della struttura IOExtSer per poter successivamente inviare un comando alle routine interne del dispositivo Serial. Alcuni di questi parametri sono specifici della struttura IOExtSer, gli altri costituiscono il consueto insieme di parametri richiesto dalla maggior parte dei dispositivi. Nella figura sono illustrati anche i flag del parametro io_SerFlags che il task imposta a seconda del modo in cui dev'essere eseguito il comando.

Tavola 6.2: *Bit del registro di stato della porta seriale*

Bit	Significato
0-1	Riservati
2	A500 e A2000: indicatore di chiamata sulla porta seriale e stampante selezionata sulla porta parallela
	A1000: indicatore di chiamata (bit = 1)
3	Periferica pronta (bit = 0)
4	OK per trasmettere (bit = 0)
5	Portante presente (bit = 0)
6	Richiesta per trasmettere (bit = 0)
7	Computer pronto (bit = 0)
8	Overflow del buffer di lettura (bit = 1)
9	Segnale di break inviato (bit = 1)
10	Segnale di break ricevuto (bit = 1)
11	Trasmette XOFF (bit = 1)
12	Ricevuto XOFF (bit = 1)
13-15	Riservati

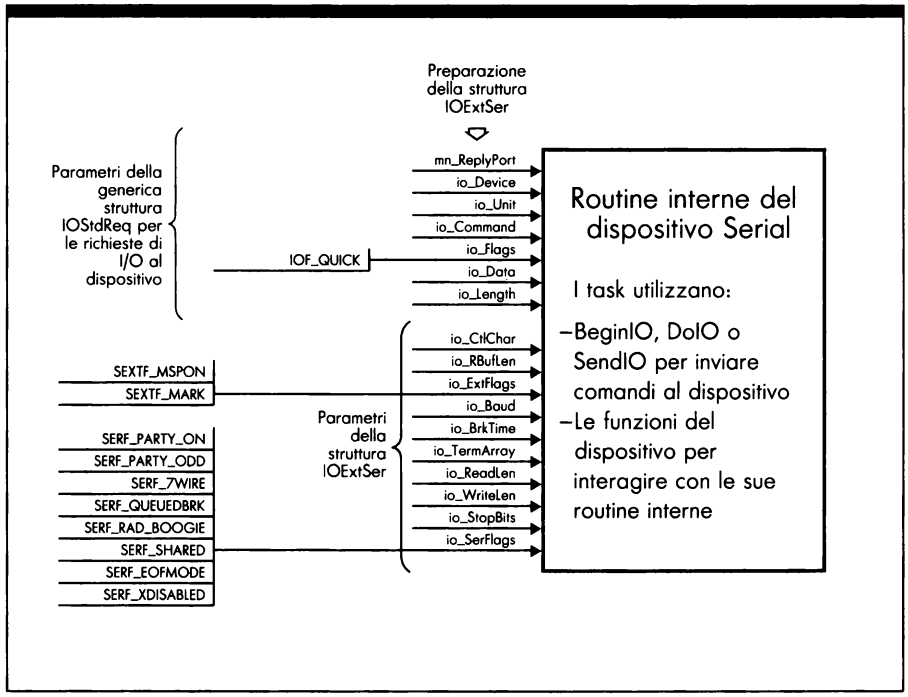


Figura 6.2a: *Gestione delle funzioni e dei comandi previsti dal dispositivo Serial (input)*

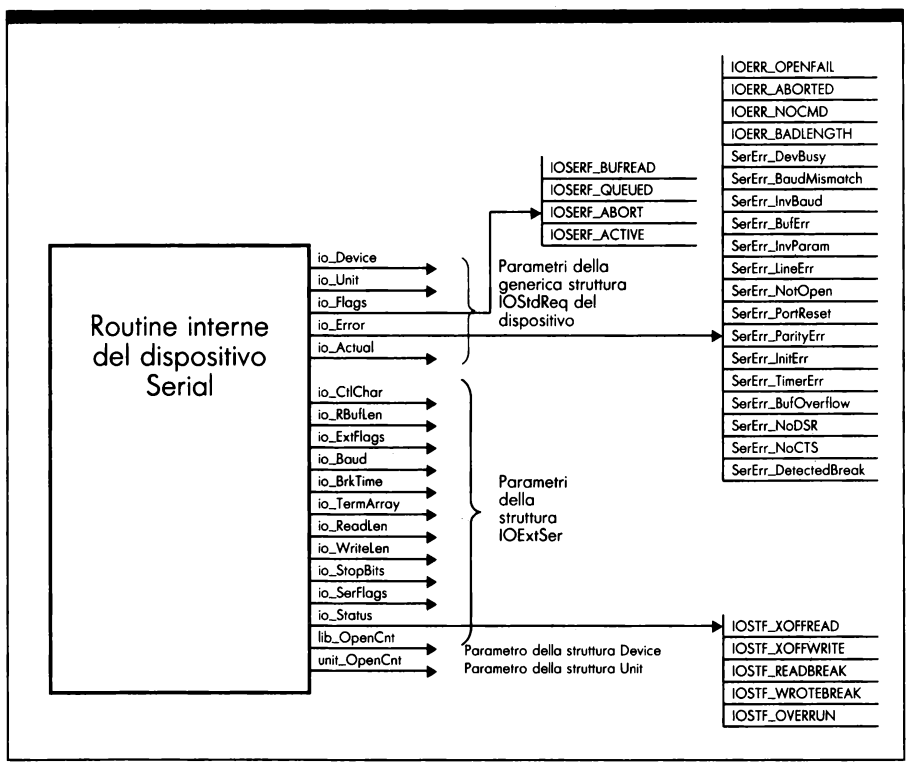


Figura 6.2b: *Gestione delle funzioni e dei comandi previsti dal dispositivo Serial (output)*

2. *Invio del comando e sua elaborazione*. In questa fase, l'unico compito del programmatore è quello d'inviare il comando al dispositivo tramite una delle funzioni BeginIO, DoIO o SendIO. In seguito il controllo passa al sistema e alle routine interne del dispositivo.

3. *Elaborazione dei parametri di output del comando e loro restituzione*. Questa fase è controllata completamente dal sistema e dalle routine interne del dispositivo. I risultati prodotti dall'elaborazione della richiesta vengono restituiti al task "mittente" nei soliti modi. Se il comando non è immediato e la richiesta non è stata inviata richiedendo il QuickIO, il dispositivo la elabora nel momento in cui raggiunge la sommità della coda alla request port e successivamente la restituisce alla reply port del task (quindi viene nuovamente accodata). Nel caso invece che sia stato richiesto il QuickIO, oppure che il comando sia immediato, non si verifica alcun accodamento alla request port e la richiesta giunge direttamente alle routine interne del dispositivo.

I risultati prodotti dall'elaborazione del comando si trovano nei parametri io_Error, io_Actual e io_Status. Questi parametri forniscono indicazioni sugli eventuali errori verificatisi, sul numero di byte trasferiti e sullo stato della richiesta.

Le Figure 6.2a e 6.2b (nella pagina precedente) descrivono inoltre i parametri più importanti nel funzionamento del dispositivo Serial. Le funzioni OpenDevice e CloseDevice modificano come sempre il parametro lib_OpenCnt contenuto nella struttura Device. OpenDevice, inoltre, influenza il parametro io_Error e può aggiornare con valori di default altri parametri della struttura IOExtSer.

Il dispositivo Serial, al pari del dispositivo Parallel, dispone di un'unica unità e quindi non impiega la struttura Unit per definirla. OpenDevice non memorizza alcun indirizzo nel parametro io_Unit della struttura di I/O, né incrementa il parametro unit_OpenCnt illustrato nei primi due capitoli del libro.

Le strutture del dispositivo Serial

Il dispositivo Serial prevede due strutture: la struttura IOExtSer e la struttura IOTArray (si veda la Figura 6.3 nella pagina successiva). La struttura IOExtSer costituisce il messaggio non standard che i task devono utilizzare per comunicare con il dispositivo. Al suo interno, come per tutte le strutture di I/O non standard, il primo elemento è la sotto-struttura IOStdReq (denominata IOSer), che costituisce l'intestazione del messaggio (struttura Message) e contiene alcuni parametri standard. Oltre a questa sotto-struttura, IOExtSer contiene anche la sotto-struttura IOTArray (denominata io_TermArray), nella quale i task possono indicare i caratteri di EOF da impiegare per la ricezione di dati.

Nelle due descrizioni che seguono, per ogni parametro viene indicato il

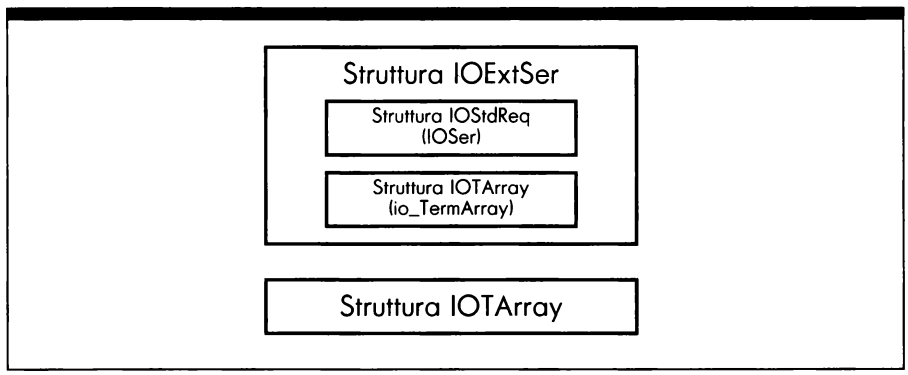


Figura 6.3: *Strutture utilizzate dal dispositivo Serial*

valore di default, cioè il valore che assume quando il dispositivo viene aperto; per alcuni parametri il valore di default è quello impostato dal programma Preferences o dal comando SDCMD_SETPARAMS.

La struttura IOTArray

La struttura IOTArray è costituita da due long word nelle quali il task memorizza in sequenza i caratteri di EOF. Finora ci siamo riferiti a questo insieme di caratteri chiamandolo array ma in realtà, dal punto di vista del linguaggio C, nella struttura IOTArray non compare nessun array. D'altra parte, quando il dispositivo riceve il controllo della struttura esamina i caratteri di EOF trattando l'insieme come un array, ed è per questo che ci riferiamo a essi come se fossero contenuti in un array.

La struttura IOTArray è definita come segue:

```
struct IOTArray {
    ULONG TermArrayØ;
    ULONG TermArray1;
};
```

I parametri della struttura IOTArray sono i seguenti:

- TermArray0. È costituito da 4 byte, nei quali il task definisce i primi quattro caratteri (qualsiasi numero esadecimale compreso tra 0x00 e 0xFF) che desidera indicare come caratteri di EOF. È assolutamente necessario che questi numeri vengano memorizzati in ordine decrescente.

- TermArray1. È costituito da 4 byte, nei quali il task definisce il secondo gruppo di 4 caratteri che desidera indicare come caratteri di EOF. Di nuovo devono essere memorizzati in ordine decrescente, anche rispetto ai valori memorizzati in TermArray0. Il sistema verifica la presenza dei

caratteri di EOF nel flusso entrante di dati solo se il task ha impostato il flag SERF_EOFMODE del parametro io_SerFlags.

Si noti che se vengono utilizzati meno di otto caratteri di EOF, occorre riempire la parte residua dell'array ripetendo il più basso codice ASCII utilizzato. Per esempio, l'array di codici ASCII x0807060504030303 definisce sei caratteri di EOF su otto disponibili; il task ha quindi riempito l'array ripetendo l'ultimo codice (0x03).

I caratteri di EOF impostati tramite il comando SDCMD_SETPARAMS vengono mantenuti internamente dal dispositivo anche dopo la sua chiusura da parte del task, fino a quando il dispositivo non viene rimosso dalla memoria, oppure non viene inviato un nuovo comando SDCMD_SETPARAMS.

La struttura IOExtSer

La struttura IOExtSer è definita come segue:

```
struct IOExtSer {
        struct IOStdReq IOSer;
        ULONG io_CtlChar;
        ULONG io_RBufLen;
        ULONG io_ExtFlags;
        ULONG io_Baud;
        ULONG io_BrkTime;
        struct IOTArray io_TermArray;
        UBYTE io_ReadLen;
        UBYTE io_WriteLen;
        UBYTE io_StopBits;
        UBYTE io_SerFlags;
        UWORD io_Status;
};
```

I parametri della struttura IOExtSer hanno i seguenti significati:

- IOSer. È la sotto-struttura di tipo IOStdReq che intesta il messaggio IOExtSer. Nell'intestazione assume particolare importanza il parametro mn_ReplyPort, che il task deve inizializzare con l'indirizzo della reply port (struttura MsgPort) a cui le richieste devono essere accodate quando vengono restituite.

- io_CtlChar. Questo parametro è costituito da quattro caratteri di controllo: XON, XOFF, INQ e ACK. I primi due si riferiscono al protocollo di trasmissione XON/XOFF, mentre gli altri al protocollo di trasmissione INQ/ACK (non previsto dalla versione 1.3 del software sistema). Per default questo parametro contiene il valore 0x11130000, che assegna il carattere Ctrl-Q a XON e Ctrl-S a XOFF. Se questo parametro viene impostato tramite il comando SDCMD_SETPARAMS, il suo contenuto

permane anche dopo la chiusura del dispositivo, fino a quando questo non viene rimosso dalla memoria oppure non viene inviato un nuovo comando SDCMD_SETPARAMS. Per maggiori informazioni si vedano le discussioni dei comandi CMD_READ e CMD_WRITE in questo capitolo.

- io_RBufLen. Indica la lunghezza (in byte) del buffer interno del dispositivo Serial. Il valore minimo che può assumere è 64. OpenDevice lo inizializza sempre con il valore impostato nel programma Preferences (per default 512 byte). Se questo parametro viene modificato e la struttura viene inviata al dispositivo indicando il comando SDCMD_SETPARAMS, il dispositivo libera la memoria occupata dal precedente buffer interno e ne alloca un nuovo della dimensione richiesta: il contenuto del precedente buffer viene completamente perso.

- io_ExtFlags. È costituito da un insieme di flag che può essere utilizzato soltanto a partire dalla versione 1.2 del software sistema. Impostando il flag SEXTF_MSPON il dispositivo Serial utilizza per le operazioni di lettura e scrittura il controllo di parità mark-space anziché odd-even. Impostando il flag SEXTF_MARK il dispositivo Serial utilizza la parità mark quando il task, impostando il flag SEXTF_MSPON, seleziona la parità mark-space. Si noti che impostando uno di questi due flag si provoca l'automatica inizializzazione del flag SERF_PARTY_ON, e si impone al dispositivo d'ignorare il flag SERF_PARTY_ODD.

- io_Baud. In questo parametro il task deve indicare la velocità di trasmissione, che può variare fra 110 e 292.000 baud. Si sconsiglia di oltrepassare i 31.250 baud, specialmente se il sistema è fortemente occupato. OpenDevice inizializza sempre questo parametro con il valore impostato nel programma Preferences (per default 9.600 baud).

- io_BrkTime. Indica la durata del segnale di break in microsecondi. Il suo valore di default è 250.000. Se questo parametro viene impostato tramite il comando SDCMD_SETPARAMS, il suo contenuto permane anche dopo la chiusura del dispositivo, fino a quando questo non viene rimosso dalla memoria oppure non viene inviato un nuovo comando SDCMD_SETPARAMS.

- io_TermArray. Costituisce il nome della struttura di tipo IOTArray che permette al task d'indicare i propri particolari caratteri di EOF. Se questo parametro viene impostato tramite il comando SDCMD_SETPARAMS, il suo contenuto permane anche dopo la chiusura del dispositivo, fino a quando questo non viene rimosso dalla memoria oppure non viene inviato un nuovo comando SDCMD_SETPARAMS.

- io_ReadLen. Indica a CMD_READ il numero di bit che compongono ogni carattere che arriva alla porta seriale. OpenDevice inizializza sempre questo parametro con il valore impostato nel programma Preferences (per default 8 bit). Si noti che se viene impostato a 7, il bit più significativo di ogni byte trasmesso dal computer collegato alla porta seriale dell'Amiga viene perso. Ciò può essere pericoloso quando si ricevono file di dati diversi dai testi e quando si indicano caratteri di EOF di valore superiore a 0x7F.

- io_WriteLen. Indica a CMD_WRITE il numero di bit che deve impiegare per ogni carattere che invia alla porta seriale. OpenDevice inizializza questo parametro con il valore impostato nel programma Preferences (per default 8 bit).

- io_StopBits. Indica il numero dei bit di stop associati a ogni carattere trasmesso o ricevuto. OpenDevice inizializza questo parametro con il valore impostato nel programma Preferences (per default 1 bit).

- io_SerFlags. Questo parametro è costituito da un insieme di flag che modificano il comportamento del dispositivo nel trasferimento dei dati. La descrizione dei singoli flag si trova nel prossimo paragrafo.

- io_Status. Questo parametro è costituito da un insieme di flag che descrivono lo stato della porta seriale.

I flag che controllano il comportamento del dispositivo Serial

I significati dei flag che il task può modificare nel parametro io_SerFlags sono i seguenti:

- SERF_XDISABLED (bit 7). Impostando questo flag si disabilita il protocollo XON/XOFF. Questo protocollo utilizza i codici ASCII DC3 (Ctrl-S) e DC1 (Ctrl-Q) per sospendere (XOFF) e riattivare (XON) il trasferimento dei dati. Se si impiega il protocollo XON/XOFF e chi sta ricevendo invia il carattere XOFF, chi trasmette sospende automaticamente la trasmissione e si pone in attesa del carattere XON. Se chi trasmette è l'Amiga e il task non ha impostato il flag SERF_XDISABLED, il dispositivo Serial controlla automaticamente se giungono alla porta seriale caratteri XON. Il task può avere la necessità di disattivare questo protocollo se deve inviare o ricevere un file di codici eseguibili (protocolli XModem e Kermit) o un'immagine, due esempi nei quali è plausibile aspettarsi il transito di byte di valore 0x11 (Ctrl-Q) e 0x13 (Ctrl-S). Si noti che il flag SERF_XDISABLED è l'unico del parametro io_SerFlags che i task possono modificare (tramite il comando SDCMD_SETPARAMS) mentre è in corso un trasferimento attraverso la porta seriale. OpenDevice imposta questo flag quando nel programma Preferences risulta selezionata l'opzione None o RTS/CTS per

l'handshaking, indipendentemente da come il task aveva impostato il parametro io_SerFlags prima di aprire il dispositivo. Il comando SDCMD_SETPARAMS permette di variare lo stato di questo flag, ma non in maniera permanente: quando il task chiude il dispositivo il flag torna allo stato indicato nel programma Preferences.

- SERF_EOFMODE (bit 6). Impostando questo flag si fa in modo che il dispositivo consideri come caratteri di EOF quelli che il task indica nella sotto-struttura io_TermArray della struttura IOExtSer. Questo flag e SERF_QUEUEDBRK possono essere alterati direttamente dal task senza inviare il comando SDCMD_SETPARAMS (la stessa possibilità esiste per i flag SERF_SHARED e SERF_7WIRE).

- SERF_SHARED (bit 5). Impostando questo flag si apre il dispositivo in modo condiviso. I task possono impostarlo prima di chiamare la funzione OpenDevice, o anche quando il dispositivo è già stato aperto, prima d'inviare il comando SDCMD_SETPARAMS. Una volta che il dispositivo è stato aperto in modo condiviso, non è possibile azzerare il flag SERF_SHARED tramite il comando SDCMD_SETPARAMS. SERF_SHARED è l'unico flag di cui il dispositivo controlla lo stato quando il task esegue la funzione OpenDevice.

- SERF_RAD_BOOGIE (bit 4). Impostando questo flag si abilita il modo di trasferimento dei dati ad alta velocità. Questa tecnica viene frequentemente utilizzata per effettuare comunicazioni attraverso un'interfaccia MIDI (Musical Instrument Digital Interface, interfaccia digitale per strumenti musicali) e comandare quindi strumenti musicali come i sintetizzatori. Con questo flag impostato il dispositivo Serial riduce al minimo le sue operazioni nel trasferimento dati: non esegue i controlli di parità, non gestisce i caratteri di XOFF, non riconosce caratteri di lunghezza superiore a 8 bit e non esegue test sul segnale di break.

  Infine, impostando il flag SERF_RAD_BOOGIE si causa l'automatica inizializzazione del bit SERF_XDISABLED. Se questo flag viene impostato tramite il comando SDCMD_SETPARAMS, il suo contenuto permane anche dopo la chiusura del dispositivo, fino a quando questo non viene rimosso dalla memoria oppure non viene inviato un nuovo comando SDCMD_SETPARAMS.

- SERF_QUEUEDBRK (bit 3). Impostando questo flag il comando SDCMD_BREAK viene accodato. Il flag dev'essere ovviamente impostato nel parametro io_SerFlags della struttura IOExtSer che il task impiega per inviare il comando. Se questo flag viene impostato tramite il comando SDCMD_SETPARAMS, il suo contenuto permane anche dopo la chiusura del dispositivo, fino a quando questo non viene rimosso dalla memoria oppure non viene inviato un nuovo comando SDCMD_SETPARAMS.

- SERF_7WIRE (bit 2). Impostando questo flag si utilizza il protocollo 7-WIRE (RS-232C RTS/CTS) per le operazione CMD_WRITE e CMD_READ. Questo protocollo prevede l'uso di due linee elettriche in più rispetto alle cinque previste dall'interfaccia seriale standard. OpenDevice imposta questo flag quando nel programma Preferences risulta selezionata l'opzione None o RTS/CTS per l'handshaking, indipendentemente da come il task aveva impostato il parametro io_SerFlags prima di aprire il dispositivo. Il comando SDCMD_SETPARAMS permette di variarne lo stato ma non in maniera permanente: quando il task chiude il dispositivo il flag torna allo stato indicato nel programma Preferences.

- SERF_PARTY_ODD (bit 1). Impostando questo flag il dispositivo effettua il controllo di parità dispari durante i trasferimenti dei dati. OpenDevice lo imposta insieme con il flag SERF_PARTY_ON (che abilita il controllo della parità) quando nel programma Preferences risulta selezionata l'opzione Odd per il controllo di parità, indipendentemente da come il task aveva impostato il parametro io_SerFlags prima di aprire il dispositivo. Il comando SDCMD_SETPARAMS permette di variare lo stato di questo flag ma non in maniera permanente: quando il task chiude il dispositivo il flag torna allo stato indicato nel programma Preferences.

- SERF_PARTY_ON (bit 0) Impostando questo flag il dispositivo effettua il controllo della parità (mark-space o odd-even) durante i trasferimenti dei dati. Se non viene impostato anche il flag SERF_PARTY_ODD il dispositivo esegue il controllo di parità pari. OpenDevice imposta questo flag senza però impostare anche il flag SERF_PARTY_ODD quando nel programma Preferences risulta selezionata l'opzione Even per il controllo di parità, indipendentemente da come il task aveva impostato il parametro io_SerFlags prima di aprire il dispositivo. Il comando SDCMD_SETPARAMS permette di variare lo stato di questo flag, ma non in maniera permanente: quando il task chiude il dispositivo il flag torna allo stato indicato nel programma Preferences.

I significati dei flag che il dispositivo può modificare nel parametro io_Flags della struttura IOStdReq sono i seguenti:

- IOSERF_BUFREAD (bit 7). Questo flag viene impostato quando il dispositivo trasferisce byte dal buffer interno del dispositivo al buffer del task.

- IOSERF_QUEUED (bit 6). Questo flag viene impostato quando la richiesta di I/O viene accodata, cioè quando non viene accolto il QuickIO.

- IOSERF_ABORT (bit 5). Questo flag viene impostato quando la richiesta di I/O è stata eliminata dalla funzione AbortIO, oppure dai comandi CMD_RESET o CMD_FLUSH.

- IOSERF_ACTIVE (bit 4). Questo flag viene impostato per indicare che l'elaborazione della richiesta non si è ancora conclusa (la richiesta può quindi trovarsi ancora nella coda alla request port, oppure essere in corso di elaborazione).

I significati dei flag che il dispositivo restituisce nel parametro io_Status quando riceve il comando SDCMD_QUERY sono i seguenti:

- IOSTF_XOFFREAD (bit 12). Il dispositivo imposta questo flag quando riceve dall'hardware collegato un carattere XOFF.

- IOSTF_XOFFWRITE (bit 11). Il dispositivo imposta questo flag dopo aver trasmesso all'hardware collegato un carattere XOFF.

- IOSTF_READBREAK (bit 10). Il dispositivo imposta questo flag quando riceve un segnale di break; il segnale di break viene utilizzato per sospendere il trasferimento dei dati per un periodo di tempo prestabilito.

- IOSTF_WROTEBREAK (bit 9). Il dispositivo imposta questo flag dopo aver trasmesso attraverso la porta seriale un segnale di break.

- IOSTF_OVERRUN (bit 8). Il dispositivo imposta questo flag quando la richiesta di I/O provoca l'overflow del buffer interno del dispositivo Serial.

I codici d'errore che il task può ricevere nel parametro io_Error delle strutture di I/O che invia al dispositivo sono i seguenti:

- IOERR_OPENFAIL, IOERR_ABORTED, IOERR_NOCMD e IOERR_BADLENGTH hanno il significato comune a tutti i dispositivi dell'Amiga (si veda il capitolo 3).

- SerErr_DevBusy. Le routine interne del dispositivo Serial sono occupate e non possono soddisfare la richiesta di I/O. Questo codice d'errore viene restituito anche nel caso che il task cerchi di aprire il dispositivo quando un altro task lo ha aperto in modo esclusivo o quando il task cerca di aprirlo in modo esclusivo e il dispositivo è già stato aperto in modo condiviso da altri task.

- SerErr_BaudMismatch. Si è verificata un'incompatibilità tra la velocità di trasmissione attuale e quella richiesta.

- SerErr_InvBaud. Il task ha indicato un valore non corretto per la velocità di trasmissione.

- SerErr_BufErr. Il dispositivo non riesce ad allocare il buffer interno.

- SerErr_InvParam. Il task ha specificato un parametro non valido nella struttura utilizzata per la richiesta di I/O.
- SerErr_LineErr. Si è verificato un errore di linea durante un trasferimento di dati; di solito questo evento segnala la presenza di un collegamento difettoso con la porta seriale.
- SerErr_ParityErr. Si è verificato un errore di parità durante il trasferimento dei dati.
- SerErr_TimerErr. Si è verificato un errore di temporizzazione durante il trasferimento dei dati.
- SerErr_BufOverflow. Si è verificato un errore di overflow nel buffer durante il trasferimento dei dati.
- SerErr_NoDSR. Non è stato rilevato il segnale DSR (data set ready), cioè la periferica collegata non è pronta.
- SerErr_DetectedBreak. È stato rilevato un segnale di break durante il trasferimento dei dati; la trasmissione viene momentaneamente sospesa.
- SerErr_NotOpen. Il task non ha aperto il dispositivo Serial prima d'inviare il comando.
- SerErr_PortReset. Il dispositivo Serial è stato reinizializzato dal comando CMD_RESET.
- SerErr_InitErr. Si è verificato un errore durante l'inizializzazione del dispositivo Serial in seguito a un comando CMD_RESET.
- SerErr_NoCTS. Non è stato rilevato il segnale CTS (clear to send) che la periferica collegata deve inviare per dare il permesso all'avvio della trasmissione.

Le condizioni di EOF

Il dispositivo Serial prevede diversi sistemi per controllare la quantità di caratteri ricevuti o trasmessi lungo la linea seriale. Iniziamo descrivendo i sistemi disponibili quando si ordina al dispositivo di effettuare un'operazione di lettura, cioè di acquisire dati provenienti dalla linea seriale.

Acquisizione di dati

Il metodo più semplice per controllare l'afflusso di dati è indicare al dispositivo Serial la quantità esatta di byte che devono essere trasferiti nel buffer di lettura del task con un comando CMD_READ. Questa quantità dev'essere memorizzata nel parametro io_Length della richiesta di I/O prima di inoltrarla. In questo modo, il dispositivo restituisce la richiesta al task quando nel suo buffer interno è arrivato il previsto numero di byte: il dispositivo non compie nessuna analisi dei dati in ingresso alla porta seriale, cioè nessun dato può interrompere l'afflusso. Questo metodo viene impiegato quando i dati possono essere di qualsiasi tipo, come nella ricezione di codici eseguibili.

Può invece essere necessario che il dispositivo Serial riconosca il codice 0x00 come carattere di EOF, cioè carattere che interrompe l'afflusso dei dati. Perché questo accada, nel parametro io_Length della richiesta di I/O dev'essere memorizzato il valore −1. In questo modo, il dispositivo Serial accetta caratteri fino a quando non rileva nel flusso il carattere 0x00. Si tratta di un metodo che presenta alcuni rischi, in quanto il dispositivo non sa quali sono i limiti di capienza del buffer messo a disposizione dal task e quindi non può rilevare un suo eventuale overflow. Quando il task riottiene la richiesta di I/O (il dispositivo ha individuato uno 0x00 nel flusso entrante), può esaminare il parametro io_Actual per sapere quanti byte sono stati ricevuti.

Oltre a questi due metodi base, il task può indicare al dispositivo fino a otto caratteri di EOF "speciali", che il dispositivo prende in considerazione se il task ha impostato il flag SERF_EOFMODE nella richiesta di I/O che definisce il comando CMD_READ. In questo caso, qualunque sia il numero contenuto nel parametro io_Length (compreso −1), se il dispositivo rileva nel flusso entrante la presenza di uno dei caratteri di EOF "speciali" sospende la ricezione e restituisce la richiesta di I/O al task.

Trasmissione di dati

Nella trasmissione il dispositivo Serial non consente d'indicare caratteri di EOF speciali. Quindi ci sono soltanto due metodi per controllare il numero di caratteri che il dispositivo invia in seguito a un comando CMD_WRITE: indicare nel parametro io_Length il numero di byte da trasferire, oppure indicare nel parametro io_Length il valore −1 e assicurarsi che il parametro io_Data della richiesta di I/O individui un buffer in memoria nel quale sia stato inserito un opportuno byte a zero per indicare la fine della trasmissione.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOExtSer)
A1

Scopo della funzione

Questa funzione chiude l'accesso da parte del task all'unica unità del dispositivo Serial, l'unità 0. CloseDevice prevede la chiusura del dispositivo Timer (aperto automaticamente durante l'esecuzione di OpenDevice), la liberazione della memoria occupata dal buffer interno del dispositivo Serial e la decrementazione del parametro lib_OpenCnt contenuto nella struttura Device del dispositivo. Se in questa fase lib_OpenCnt arriva a zero e il flag LIBF_DELEXP del parametro lib_Flags nella struttura Device risulta impostato, il dispositivo Serial viene eliminato dalla memoria.

La funzione CloseDevice provvede inoltre ad aggiornare con il valore −1 il puntatore io_Device della sotto-struttura IOStdReq, e con il valore 0 il puntatore io_Unit. Dopo l'esecuzione di CloseDevice, se il task vuole nuovamente accedere al dispositivo deve chiamare ancora la funzione OpenDevice.

Argomenti della funzione

iOExtSer — Rappresenta l'indirizzo della struttura di tipo IOExtSer che il task impiega per interagire con il dispositivo Serial. Generalmente questa struttura viene parzialmente inizializzata dalla funzione OpenDevice quando il task apre il dispositivo. Si noti che comunque questo indirizzo può anche indicare una struttura diversa da quella utilizzata con OpenDevice, a patto però che i parametri io_Device e io_Unit siano comunque quelli restituiti da OpenDevice.

Discussione

CloseDevice permette a un task di terminare l'accesso al dispositivo Serial. È assolutamente indispensabile che il task chiuda il dispositivo quando non ne ha più bisogno, soprattutto se l'ha aperto in modo esclusivo. Prima di chiamare CloseDevice deve però verificare che tutte le richieste di I/O inviate abbiano ricevuto la relativa risposta dalle routine interne del dispositivo Serial. È possibile effettuare questa operazione utilizzando le funzioni GetMsg, CheckIO e WaitIO, le quali verificano, in modi diversi, la presenza di richieste nella coda alla reply port del task.

Si noti infine che i parametri io_CtlChar, io_BrkTime, io_TermArray, e alcuni flag del parametro io_SerFlags vengono conservati dal dispositivo anche dopo l'esecuzione della funzione CloseDevice se sono stati impostati tramite il comando SDCMD_SETPARAMS.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("serial.device", ØL, iOExtSer, ØL)
DØ                 AØ                DØ A1       D1
```

Scopo della funzione

Questa funzione apre l'accesso all'unità 0 del dispositivo Serial e provvede anche ad aprire il dispositivo Timer. Il task deve indicare come argomento l'indirizzo di una struttura di tipo IOExtSer (che ha precedentemente allocato in memoria) nella quale la funzione OpenDevice inizializza i parametri di gestione comuni a tutti i dispositivi, come io_Device e io_Unit, e alcuni parametri tipici del dispositivo Serial. Questi parametri specifici sono quelli che descrivono le caratteristiche delle comunicazioni che il task avvia tramite i comandi CMD_READ e CMD_WRITE, e vengono aggiornati dalla funzione OpenDevice con i loro valori di default.

Il dispositivo Serial può essere aperto sia in modo condiviso sia in modo esclusivo. Se al momento della chiamata a OpenDevice il dispositivo non risulta in memoria, il sistema provvede a caricarlo da disco e ad allocarlo in memoria.

I parametri restituiti dalla funzione sono i seguenti:

- io_Device. Viene inizializzato con l'indirizzo della struttura Device utilizzata per gestire il dispositivo Serial.

- io_Unit. Viene restituito azzerato, dal momento che il dispositivo Serial è in grado di gestire una sola porta seriale. Per ragioni di compatibilità, il valore zero dev'essere indicato ogni volta che si invia un comando al dispositivo.

- io_Error. Un valore di questo parametro pari a 0 indica che l'apertura ha avuto successo. IOERR_OPENFAIL indica che non è stato possibile aprire il dispositivo Serial (per esempio perché non si trovava nella directory logica DEVS: o se non c'era sufficiente memoria per allocarlo). Se si tenta di aprire il dispositivo nel modo di accesso esclusivo quando è già stato aperto da altri task, oppure se si tenta di aprirlo in modo condiviso quando un task lo detiene in modo esclusivo, viene restituito il codice d'errore SerErr_DevBusy.

- io_CtlChar. OpenDevice memorizza in questo parametro i caratteri di XON (0x11) e di XOFF (0x13) adottati per default nelle trasmissioni che prevedono il protocollo XON/XOFF. Gli altri due byte della long word vengono azzerati. I valori di default possono essere anche quelli impostati da una precedente esecuzione del comando SDCMD_SETPARAMS.

- io_RBufLen. OpenDevice memorizza in questo parametro il numero impostato dal programma Preferences, o quello impostato dall'ultima esecuzione del comando SDCMD_SETPARAMS, che rappresenta la grandezza del buffer interno del dispositivo.

- io_Baud. OpenDevice memorizza in questo parametro il numero impostato dal programma Preferences o quello impostato dall'ultima esecuzione del comando SDCMD_SETPARAMS.

- io_BrkTime. OpenDevice memorizza in questo parametro il valore 250.000, tempo di break di default, o il valore impostato dall'ultima esecuzione del comando SDCMD_SETPARAMS.

- io_TermArray. OpenDevice memorizza in questo parametro i caratteri di EOF impostati dall'ultima esecuzione del comando SDCMD_SETPARAMS.

- io_ReadLen. OpenDevice memorizza in questo parametro il valore impostato dal programma Preferences o il valore impostato dall'ultima esecuzione del comando SDCMD_SETPARAMS, che indica la lunghezza dei byte impiegati in ricezione.

- io_WriteLen. OpenDevice memorizza in questo parametro il valore impostato dal programma Preferences o il valore impostato dall'ultima esecuzione del comando SDCMD_SETPARAMS, che indica la lunghezza dei byte impiegati in trasmissione.

- io_StopBits. OpenDevice memorizza in questo parametro il valore impostato dal programma Preferences, o il valore impostato dall'ultima esecuzione del comando SDCMD_SETPARAMS, che indica il numero di bit di stop che il dispositivo accetta in trasmissione.

- io_SerFlags. OpenDevice imposta in questo parametro alcuni flag che dipendono dai valori riportati in Preferences o impostati dall'ultima esecuzione del comando SDCMD_SETPARAMS. I flag che vengono influenzati dal programma Preferences sono SERF_XDISABLED, SERF_7WIRE, SERF_PARTY_ON, SERF_PARTY_ODD. I flag che vengono influenzati dai valori impostati con l'ultima esecuzione del comando SDCMD_SETPARAMS sono SERF_EOFMODE, SERF_RAD_BOOGIE, SERF_QUEUEDBRK. Il flag SERF_SHARED risulta impostato soltanto se il task l'ha impostato prima di chiamare OpenDevice e il dispositivo non risulta aperto in maniera esclusiva da un altro task.

Argomenti della funzione

"serial.device" Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Serial.

ØL Questo argomento indica che si desidera aprire l'unità 0 del dispositivo, l'unica disponibile.

iOExtSer Questo argomento dev'essere l'indirizzo della struttura di tipo IOExtSer che il task intende impiegare per interagire con il dispositivo.

ØL Questo valore indica che l'argomento flag non viene preso in considerazione dalla funzione.

Preparazione della struttura IOExtSer

Per aprire il dispositivo Serial occorre inizializzare il parametro mn_ReplyPort della struttura di I/O con l'indirizzo della struttura MsgPort che rappresenta la reply port del task. Il task può allocare una message port tramite la funzione CreatePort di supporto alla libreria Exec e indicarne l'indirizzo come argomento della funzione CreateExtIO. Chiamando quest'ultima funzione il task alloca la struttura di I/O necessaria per interagire con il dispositivo e memorizza automaticamente l'indirizzo della reply port nel parametro mn_ReplyPort. Si deve inoltre impostare il flag SERF_SHARED se si vuole aprire il dispositivo Serial nel modo di accesso condiviso.

Discussione

La funzione OpenDevice viene utilizzata per mettere le routine del dispositivo Serial a disposizione di un task. Una volta in possesso del dispositivo Serial, il task può inviare una serie di comandi CMD_WRITE e CMD_READ (tramite BeginIO, DoIO oppure SendIO) per scambiare informazioni con qualsiasi dispositivo hardware esterno collegato alla porta seriale dell'Amiga. Quando tutte le operazioni di scrittura e lettura sono state portate a termine, il task dovrebbe chiudere il dispositivo Serial, soprattutto se l'ha aperto in modo esclusivo.

La funzione OpenDevice inizializza con valori di default tutti i parametri che intervengono nelle comunicazioni attraverso la porta seriale. Il task può cambiarli secondo le proprie esigenze tramite il comando SDCMD_SETPARAMS.

COMANDI STANDARD DEL DISPOSITIVO

CMD_CLEAR

Scopo del comando

CMD_CLEAR serve per azzerare i buffer interni dei dispositivi. Il dispositivo Serial possiede solo un buffer di lettura e quindi CMD_CLEAR agisce esclusivamente su questo buffer.

CMD_CLEAR viene sempre eseguito in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. SerErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtSer utilizzata per il comando CMD_CLEAR. SerErr_NotOpen indica che il dispositivo Serial non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando CMD_CLEAR.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la

funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice (zero). Il task deve impostare io_Command con il comando CMD_CLEAR e azzerare il parametro io_Flags.

Discussione

CMD_CLEAR provvede a cancellare il contenuto dell'unico buffer interno del dispositivo Serial, quello di lettura. Dopo quest'operazione, aggiorna il puntatore interno al buffer perché punti nuovamente all'inizio.

CMD_FLUSH

Scopo del comando

CMD_FLUSH elimina tutte le richieste di I/O accodate, mentre non influenza le richieste di I/O CMD_READ e CMD_WRITE in esecuzione. CMD_FLUSH viene sempre eseguito in modo immediato. Tutte le richieste di I/O eliminate vengono restituite alla reply port del task con il flag IOERR_ABORTED del parametro io_Error impostato.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. SerErr_InvParam indica che il task ha indicato un parametro non corretto nella struttura IOExtSer utilizzata per definire il comando. SerErr_NotOpen indica che il dispositivo Serial non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice (zero). Il task deve impostare io_Command con il comando CMD_FLUSH e azzerare il parametro io_Flags.

Discussione

Il comando CMD_FLUSH elimina tutte le richieste di I/O accodate alla request port dell'unità 0. Viene sempre eseguito in modo immediato. Dato che CMD_FLUSH è un comando distruttivo, dev'essere utilizzato soltanto nel caso in cui si desideri riportare il dispositivo allo stato iniziale, con la coda alla request port completamente vuota.

CMD_READ

Scopo del comando

Il comando CMD_READ ordina al dispositivo Serial di trasferire byte, nel numero indicato da io_Length, dal suo buffer interno al buffer del task, il cui indirizzo dev'essere indicato nel puntatore io_Data. Si tratta dei dati che passando attraverso il registro dati della porta seriale sono stati trasferiti nel buffer di lettura del dispositivo.

Se nel parametro io_Length viene indicato −1, il dispositivo Serial continua a leggere caratteri fino a quando non viene rilevata una condizione di EOF, che può essere l'arrivo di un byte a zero oppure di un carattere di EOF specificato dal task nella struttura IOTArray.

Se invece nel parametro io_Length viene memorizzato un numero positivo, questo rappresenta il numero di caratteri che devono essere ricevuti: il dispositivo Serial continua a leggere caratteri fino a quando non ha raggiunto la quantità indicata, anche se nel flusso entrante transitano degli zeri. In questo caso l'acquisizione dei dati può essere interrotta solo se il task ha impostato il flag SERF_EOFMODE e nel flusso arriva uno dei caratteri di EOF speciali che il task ha indicato nella struttura IOTArray.

I risultati prodotti dall'esecuzione del comando vengono restituiti nei parametri io_Actual e io_Error. Il parametro io_Actual indica il numero di caratteri effettivamente trasferiti dal buffer interno al buffer del task, che può differire dal numero indicato in io_Length se si è verificata una condizione di EOF: nel caso sia giunto un carattere di EOF, il numero restituito in io_Actual tiene conto anche della presenza del carattere di EOF (a meno che non si tratti del valore di EOF standard, cioè zero). Se si verifica un errore, nel parametro io_Error viene restituito un codice diverso da zero. I codici d'errore possibili sono i seguenti:

- SerErr_DevBusy. Le routine interne del dispositivo Serial erano occupate e non hanno potuto eseguire il comando.

- SerErr_InvParam. Il task ha specificato un parametro non valido nella struttura IOExtSer utilizzata per definire CMD_READ.

- SerErr_LineErr. Si è verificato un errore di linea durante l'operazione di lettura; di solito questo indica un difetto nella connessione elettrica tra l'hardware esterno e la porta seriale dell'Amiga.

- SerErr_NotOpen. Il dispositivo Serial non è stato aperto dal task, il quale deve quindi eseguire OpenDevice e impartire nuovamente il comando CMD_READ.

- SerErr_BufOverflow. Il buffer definito dal task è entrato in condizione di overflow; il task deve aumentare la grandezza del buffer e impartire nuovamente il comando CMD_READ.

- SerErr_BaudMismatch. La velocità di trasmissione impiegata dal dispositivo non corrisponde a quella impiegata dalla periferica collegata alla porta seriale; il task deve modificarla e impartire nuovamente il comando CMD_READ.

- SerErr_InvBaud. La velocità di trasmissione richiesta dal task non è valida (di solito perché esce dai limiti); il task deve cambiarla e impartire nuovamente il comando CMD_READ.

- SerErr_BufErr. Si è verificato un errore nel buffer interno del dispositivo durante un trasferimento di dati; il task deve individuare la causa dell'errore e impartire nuovamente il comando CMD_READ.

- SerErr_ParityErr. Si è verificato un errore di parità durante un trasferimento di dati; il task deve impartire nuovamente il comando CMD_READ.

- SerErr_TimeErr. Si è verificato un errore di sincronizzazione durante il trasferimento dei dati; il task deve impartire nuovamente il comando CMD_READ.

- SerErr_NoDSR. Non è stato rilevato il segnale DSR (data set ready) durante il trasferimento dei dati; il task deve identificare la causa e impartire nuovamente il comando CMD_READ.

- SerErr_NoCTS. Non è stato rilevato il segnale CTS (clear to send) durante il trasferimento dei dati; il task deve identificare la causa e impartire nuovamente il comando CMD_READ.

- SerErr_DetectedBreak. Il sistema ha rilevato un segnale d'interruzione durante il trasferimento dei dati; il task deve eliminare il segnale d'interruzione e impartire nuovamente il comando CMD_READ.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice (zero). Il task deve impostare io_Command con il comando CMD_READ e inizializzare i seguenti parametri:

- io_Flags. Inizializzando questo parametro a IOF_QUICK si richiede il QuickIO. In tutti gli altri casi si deve inizializzarlo a 0.

- io_SerFlags. Impostando il flag SERF_EOFMODE si richiede che il comando CMD_READ riconosca gli otto caratteri di EOF che il task ha indicato nella sotto-struttura IOTArray.

- io_Length. Si deve inizializzare questo parametro con il numero di caratteri da trasferire al buffer del task. Se il task indica il valore −1, il comando arresta la trasmissione dei dati quando rileva un byte a zero nel flusso.

- io_Data. Si deve inizializzare questo puntatore con l'indirizzo del buffer di lettura del task, nel quale il dispositivo trasferisce il numero di caratteri indicato da io_Length. Si tenga sempre presente che se il buffer del task è di dimensioni inferiori al numero di byte che vengono trasferiti, il dispositivo continua a scrivere nella RAM contigua che non è stata allocata, con conseguenze negative per l'intero sistema.

Discussione

CMD_READ permette a un task di trasferire all'interno di un proprio buffer un insieme di dati che sono giunti alla porta seriale e che il dispositivo Serial ha memorizzato nel suo buffer interno. Il buffer interno del dispositivo Serial è in pratica un parcheggio temporaneo per i dati destinati al buffer del task.

L'acquisizione viene interrotta dall'arrivo di un byte a zero se nel parametro io_Length è stato indicato il valore −1; altrimenti, i caratteri a zero non vengono considerati.

Se è stato impostato il flag SERF_EOFMODE del parametro io_SerFlags, il trasferimento dei dati viene interrotto quando il dispositivo rileva uno dei caratteri di EOF speciali definiti nella struttura IOTArray. Questo sistema permette al task di configurare le proprie operazioni di lettura nel modo più adatto alla periferica collegata. Per esempio, se la periferica utilizza il carattere Ctrl-Z per separare i blocchi di dati che invia, il task può definire un array di caratteri EOF composto da otto caratteri Ctrl-Z, di modo che il comando CMD_READ consideri concluso il trasferimento al termine di ogni blocco di dati.

È buona norma che il task impartisca un comando SDCMD_QUERY prima del comando CMD_READ, al fine di sapere quanti byte sono presenti nel buffer interno del dispositivo. Se il task richiede più byte di quanti sono effettivamente disponibili, il dispositivo non restituisce la richiesta di I/O fino a quando non riesce a trasferire il quantitativo richiesto. Se il buffer interno del dispositivo non contiene più byte da trasferire e il task desidera essere avvisato non appena ne giunge uno, conviene inviare un comando CMD_READ indicando il trasferimento di un solo byte. Quando il task riceve risposta, invia SDCMD_QUERY per sapere quanti byte sono disponibili e invia un nuovo comando CMD_READ.

CMD_RESET

Scopo del comando

CMD_RESET riporta l'intero dispositivo alle condizioni iniziali. Questa operazione comporta l'esecuzione del comando CMD_FLUSH e del comando CMD_START, che riattiva l'unità qualora sia stata bloccata da un comando CMD_STOP. Tutte le richieste di I/O CMD_READ e CMD_WRITE sia attive sia accodate alla request port vengono eliminate. Il buffer interno del dispositivo Serial viene eliminato (la RAM occupata viene liberata) e al suo posto viene allocato e inizializzato un nuovo buffer da 512 byte (la dimensione di default).

CMD_RESET viene sempre eseguito in modo immediato. Tutte le richieste di I/O eliminate vengono restituite nella coda alla reply port del task con il flag IOERR_ABORTED del parametro io_Error impostato.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. SerErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtSer utilizzata per il comando. SerErr_NotOpen indica che il dispositivo Serial non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ha ottenuto chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_RESET e azzerare il parametro io_Flags.

Discussione

CMD_RESET è un comando che ha effetti distruttivi. Tutti i dati contenuti nel buffer interno del dispositivo vengono infatti completamente persi. Inoltre, CMD_RESET chiama indirettamente CMD_START per riattivare l'unità qualora sia stata precedentemente bloccata con il comando CMD_STOP. In questo modo, quando un task inizia nuovamente a inviare richieste di I/O al dispositivo, non deve preoccuparsi di riattivarlo.

CMD_START

Scopo del comando

Se l'unità del dispositivo era stata precedentemente bloccata con il comando CMD_STOP, CMD_START la riattiva. La riattivazione riguarda qualsiasi comando CMD_READ o CMD_WRITE interrotto in fase di elaborazione. Inoltre, il comando CMD_START ordina al dispositivo Serial di riprendere la gestione della coda alla sua request port: le richieste di I/O ivi contenute riprendono la loro ascesa verso la cima della coda. CMD_START azzera sempre il parametro io_error della struttura IOExtSer.

CMD_START esegue queste azioni riattivando la sequenza di handshaking della porta seriale. Un carattere XON viene inviato al dispositivo hardware collegato, mentre un carattere logico XON viene inviato al task che si occupa del trasferimento dei dati. CMD_START viene sempre eseguito in modo immediato.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_START e azzerare il parametro io_Flags.

Discussione

CMD_START riattiva le operazioni di scrittura e lettura dei dati

precedentemente bloccate tramite il comando CMD_STOP. Questo comando è simile al comando Ctrl-Q utilizzato per far ripartire l'output su schermo nella maggior parte dei computer. CMD_START fa ovviamente riprendere anche l'ascesa dei comandi accodati.

Il comando CMD_START invia un carattere XON al dispositivo hardware collegato alla porta seriale dell'Amiga e un carattere logico XON al task che ha originariamente inviato il comando CMD_READ o CMD_WRITE. Entrambi i comandi CMD_STOP e CMD_START lavorano soltanto se è abilitato il protocollo XON/XOFF. Questo protocollo di handshaking è quello di default, e può essere disabilitato soltanto impostando il flag SERF_XDISABLED del parametro io_SerFlags appartenente alla struttura IOExtSer.

CMD_STOP

Scopo del comando

CMD_STOP blocca l'esecuzione di un comando CMD_WRITE o CMD_READ in esecuzione nell'unità e quindi anche l'ascesa delle richieste di I/O nella coda alla request port. Il sistema continua ad accodare tutte le richieste di I/O che giungono al dispositivo, ma l'unità non le elabora fino a quando il task non invia un comando CMD_START. CMD_STOP azzera sempre il parametro io_Error della struttura IOExtSer e viene sempre eseguito in modo immediato.

CMD_STOP compie queste funzioni interrompendo la sequenza di handshaking per la porta seriale, cioè inviando un carattere XOFF all'hardware esterno e un carattere logico XOFF al task. CMD_STOP viene sempre eseguito in modo immediato.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_STOP e azzerare il parametro io_Flags.

Discussione

Il comando CMD_STOP blocca l'esecuzione di un comando CMD_READ o CMD_WRITE alla prima occasione. Questo comando è simile al comando Ctrl-S utilizzato per bloccare l'output sullo schermo nella maggior parte dei computer.

Il comando CMD_STOP invia un carattere XOFF al dispositivo hardware collegato alla porta seriale. Inoltre, invia un carattere logico XOFF al task che aveva inviato il comando CMD_READ o CMD_WRITE. Sia il comando CMD_STOP sia CMD_START funzionano soltanto se il protocollo XON/XOFF è abilitato. Questo modo di handshaking è quello di default e può essere disabilitato soltanto impostando il flag SERF_XDISABLED del parametro io_SerFlags appartenente alla struttura IOExtSer.

Non si confonda la funzione del comando CMD_STOP con quella dell'array che definisce i caratteri di EOF. CMD_STOP è infatti progettato per permettere a un task di sospendere in qualsiasi momento l'esecuzione del comando CMD_READ o CMD_WRITE.

CMD_WRITE

Scopo del comando

CMD_WRITE provoca la trasmissione di un flusso di caratteri dal buffer del task al registro dati del dispositivo Serial e quindi alla periferica collegata alla porta seriale. Il task deve indicare nel parametro io_Length della struttura IOExtSer il numero di caratteri da trasmettere; se viene indicato −1, il dispositivo Serial continua a trasmettere caratteri fino a quando non rileva nel buffer del task il carattere di EOF 0x00.

CMD_WRITE può essere trattato come una richiesta di I/O sincrono o asincrono. Se un comando CMD_WRITE è stato inviato richiedendo il QuickIO e non è stato possibile eseguirlo in quanto tale, viene normalmente accodato.

I risultati prodotti dal comando vengono indicati nei parametri io_Actual e io_Error. Il parametro io_Actual indica il numero di caratteri trasferiti. Questo valore può differire da quello che il task ha indicato nel parametro io_Length solo se si verifica un errore di trasmissione.

Se nel corso dell'operazione non si verificano errori, il parametro io_Error viene restituito azzerato. Gli altri possibili valori che può assumere hanno i seguenti significati.

- SerErr_DevBusy. Le routine interne del dispositivo Serial erano occupate e non hanno potuto eseguire il comando CMD_WRITE.

- SerErr_InvParam. Un task ha specificato un parametro non valido nella struttura IOExtSer utilizzata per CMD_WRITE.

- SerErr_LineErr. Si è verificato un errore di linea durante l'operazione di lettura; di solito questo indica un collegamento difettoso tra l'hardware esterno e la porta seriale dell'Amiga.

- SerErr_NotOpen. Il dispositivo Serial non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando CMD_WRITE.

- SerErr_InvBaud. La velocità di trasmissione richiesta dal task non è valida (in genere esce dai limiti); il task deve cambiarla prima d'impartire nuovamente il comando CMD_WRITE.

- SerErr_ParityErr. Si è verificato un errore di parità durante un trasferimento di dati; il task deve impartire nuovamente il comando CMD_WRITE.

- SerErr_TimeErr. Si è verificato un errore di sincronizzazione durante il trasferimento di dati; il task deve impartire nuovamente il comando CMD_WRITE.

- SerErr_NoDSR. Non è stato inviato il segnale DSR (data set ready) durante il trasferimento dei dati; il task deve identificarne il motivo prima d'impartire nuovamente il comando CMD_WRITE.

- SerErr_NoCTS. Non è stato inviato il segnale CTS (clear to send) durante il trasferimento dei dati; il task deve identificarne il motivo prima d'impartire nuovamente il comando CMD_WRITE.

- SerErr_DetectedBreak. Il sistema ha rilevato un segnale d'interruzione durante il trasferimento dei dati; il task deve eliminare il segnale d'interruzione prima d'impartire nuovamente il comando CMD_WRITE.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ha ottenuto chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_WRITE e inizializzare i seguenti parametri:

- io_Command. Si deve inizializzare questo parametro a CMD_WRITE.

- io_Flags. Si deve inizializzare questo parametro a IOF_QUICK per richiedere il QuickIO; altrimenti inizializzarlo a 0.

- io_Length. Si deve inizializzare questo parametro con il numero di caratteri da inviare alla porta seriale dell'Amiga, oppure inizializzarlo a −1 per indicare al task di continuare a inviare caratteri finché non viene trasferito il carattere 0x00.

- io_Data. Si deve inizializzare questo parametro in modo che punti al buffer del task contenente i caratteri da inviare alla periferica attraverso il registro dati della porta seriale.

Discussione

CMD_WRITE permette a un task d'inviare caratteri al registro dati della porta seriale e quindi a un dispositivo hardware a essa collegato, prelevandoli da un buffer che il task ha opportunamente predisposto. I dati vengono trasferiti un byte alla volta. Se il task ha indicato nel parametro io_Length il numero di caratteri da trasferire, il dispositivo trasferisce l'esatta quantità di byte indicati senza compiere nessuna analisi sui singoli byte. Se invece il task ha indicato il valore −1 nel parametro io_Length, il dispositivo continua a scrivere caratteri fino a quando il sistema non rileva nel buffer del task il carattere 0x00, che comunque viene anch'esso trasmesso.

Il buffer di input del dispositivo Serial viene utilizzato per l'esecuzione del comando CMD_READ, ma non per il comando CMD_WRITE. Con quest'ultimo, infatti, i dati contenuti nel buffer del task passano attraverso il registro dati del dispositivo Serial e giungono direttamente al dispositivo hardware esterno.

COMANDI SPECIFICI DEL DISPOSITIVO

SDCMD_BREAK

Scopo del comando

Il comando SDCMD_BREAK invia un segnale d'interruzione al dispositivo hardware esterno collegato alla porta seriale. Il segnale è in pratica un abbassamento della tensione elettrica sulla linea per un tempo prolungato. Il dispositivo invia il segnale d'interruzione inizializzando il bit UARTBRK nel

registro hardware ADKCON. Allo scadere del periodo di tempo prestabilito, il bit UARTBRK viene azzerato e il segnale d'interruzione viene disattivato. La durata del segnale di break è per default 250.000 microsecondi, ma il task può modificarla inviando il comando SDCMD_SETPARAMS con il nuovo valore nel parametro io_BrkTime.

Se il flag SERF_QUEUEDBRK del parametro io_SerFlags risulta impostato, la richiesta SDCMD_BREAK viene accodata alla request port del dispositivo e viene quindi elaborata quando ne raggiunge la sommità. Se invece il flag SERF_QUEUEDBRK non è stato impostato, il segnale d'interruzione agisce immediatamente.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. SerErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtSer utilizzata per definire il comando. SerErr_NotOpen indica che il dispositivo Serial non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ha ottenuto chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando SDCMD_BREAK e azzerare il parametro io_Flags.

Discussione

SDCMD_BREAK permette a un task d'inviare un segnale d'interruzione a un dispositivo hardware esterno attraverso la linea seriale. L'effettiva durata del segnale d'interruzione è definita dal parametro io_BrkTime della struttura IOExtSer, il quale può essere impostato utilizzando il comando SDCMD_SETPARAMS, oppure tramite una semplice istruzione di assegnazione. Una volta che il parametro io_BrkTime è stato impostato, controlla la durata del segnale d'interruzione per tutti i comandi SDCMD_BREAK che vengono inviati, fino a quando non viene inserito un nuovo valore.

Se il flag SERF_QUEUEDBRK del parametro io_SerFlags risulta impostato, i comandi SDCMD_BREAK vengono accodati alla request port del dispositivo e vengono eseguiti nell'ordine in cui giungono alla sommità. Se invece il flag SERF_QUEUEDBRK risulta azzerato, il comando SDCMD_BREAK viene eseguito in modo immediato, interrompendo la richiesta di I/O in corso di elaborazione. Quando il comando ha terminato la propria esecuzione l'interruzione viene eliminata, e l'elaborazione riprende.

Due flag nel parametro io_Status della struttura IOExtSer riportano lo stato

del segnale d'interruzione. Se il flag IOSTF_WROTEBREAK (bit 9) risulta impostato, significa che è stato inviato un segnale d'interruzione; viceversa, se il flag IOSTF_READBREAK (bit 10) risulta impostato, significa che è stato ricevuto un segnale d'interruzione. Un task può sempre esaminare questi due flag per sapere quali sono le condizioni dei segnali d'interruzione nel sistema. I task devono coordinare in modo opportuno i comandi SDCMD_BREAK con gli altri comandi e funzioni del dispositivo Serial. Bisogna infatti tener presente che SDCMD_BREAK può interagire con AbortIO, CMD_FLUSH, CMD_STOP e CMD_START.

SDCMD_QUERY

Scopo del comando

Il comando SDCMD_QUERY permette a un task di rilevare lo stato delle routine interne del dispositivo Serial e della periferica connessa alla porta seriale (permette per esempio di verificare se le routine del dispositivo Serial stanno effettuando un'operazione di lettura o scrittura sul dispositivo hardware esterno connesso alla porta seriale). Il parametro io_Status della struttura IOExtSer viene aggiornato a mano a mano che le condizioni dell'hardware e del software cambiano.

SDCMD_QUERY è un comando immediato. I risultati prodotti dall'esecuzione del comando vengono restituiti nel parametro io_Status, mentre il parametro io_Error viene sempre azzerato. I bit del parametro io_Status della struttura IOExtSer sono illustrati nella Tavola 6.2 (a pagina 202).

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando SDCMD_QUERY e azzerare il parametro io_Flags.

Discussione

Il comando SDCMD_QUERY è stato previsto per permettere a un task di controllare in qualsiasi momento le attività interne del dispositivo Serial e

dell'hardware esterno. Viene normalmente utilizzato per controllare il comportamento di un modem collegato alla porta seriale e pilotato dai comandi CMD_WRITE e CMD_READ provenienti dal task. In questo caso, infatti, il task deve conoscere le condizioni del modem: se sta lavorando con il task, se è occupato nella trasmissione o nella ricezione di dati oppure se si è verificato un errore nel trasferimento dei dati. Il parametro io_Status della struttura IOExtSer dà informazioni sullo stato del trasferimento dati tra task e modem. Inoltre, il task deve sapere se il dispositivo Serial sta effettuando un'operazione di lettura oppure un'operazione di scrittura.

SDCMD_SETPARAMS

Scopo del comando

SDCMD_SETPARAMS permette a un task di modificare i parametri che intervengono nelle comunicazioni. Viene elaborato dal dispositivo solo se non risultano attive o accodate richieste CMD_READ o CMD_WRITE. L'unica eccezione a questa regola si verifica quando il task invia il comando SDCMD_SETPARAMS per cambiare il protocollo di handshaking, abilitando o disabilitando il protocollo XON/XOFF tramite il flag SERF_XDISABLED del parametro io_SerFlags.

SDCMD_SETPARAMS è un comando immediato che in genere viene utilizzato quando un task, prima d'inviare un comando CMD_READ o CMD_WRITE, deve cambiare i parametri interni del dispositivo che corrispondono a quelli della struttura di I/O non standard IOExtSer.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. SerErr_InvParam indica che il task ha specificato un parametro non corretto nella struttura IOExtSer utilizzata per il comando. SerErr_NotOpen indica che il dispositivo Serial non è stato aperto dal task, il quale deve quindi eseguire OpenDevice prima d'impartire nuovamente il comando.

Preparazione della struttura IOExtSer

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. Il parametro io_Device deve contenere l'indirizzo della struttura Device che il task ottiene chiamando la funzione OpenDevice. Il parametro io_Unit deve contenere il valore restituito dalla funzione OpenDevice. Il task deve impostare io_Command con il comando SDCMD_SETPARAMS, azzerare il parametro io_Flags e inizializzare i parametri che seguono.

- io_CtlChar. Si devono inizializzare i primi due byte di questo parametro con i valori dei caratteri XON e XOFF (si ricordi che i caratteri di controllo del protocollo INQ/ACK, gli altri due byte, attualmente non vengono impiegati). Il comando rende permanenti i nuovi caratteri, anche dopo la chiusura del dispositivo.

- io_RBufLen. Si deve inizializzare questo parametro con la dimensione del buffer interno del dispositivo Serial (si rammenti che questo *non* è il buffer definito dal task attraverso il parametro io_Data). Il buffer interno dev'essere ampio almeno 64 byte. Qualsiasi cambiamento nel parametro io_RBufLen provoca l'eliminazione del contenuto del buffer e l'allocazione, con inizializzazione, di un nuovo buffer. Quindi, il contenuto del buffer precedente viene perso. Il comando rende permanente la nuova dimensione del buffer, anche dopo la chiusura del dispositivo.

- io_ExtFlags. Si deve inizializzarlo a 0, oppure impostare opportunamente i flag SEXTF_MSPON se si desidera che il dispositivo Serial utilizzi il controllo di parità mark-space invece che odd-even per le operazioni di lettura e scrittura. Si deve impostare il flag SEXTF_MARK se si desidera che il dispositivo Serial utilizzi la parità mark quando il task, impostando il flag SEXTF_MSPON, seleziona la parità mark-space. Si noti che impostando uno di questi due flag si provoca l'automatica inizializzazione del flag SERF_PARTY_ON, e s'impone al dispositivo d'ignorare il flag SERF_PARTY_ODD.

- io_Baud. Si deve inizializzare questo parametro con la velocità di trasmissione desiderata. I limiti ammessi sono da 110 a 292.000 baud, estremi compresi. Però, anche se le capacità dell'hardware permettono di arrivare a 292.000 baud, nel caso di I/O asincrono un trasferimento di dati a una velocità superiore a 31.250 baud può non essere eseguito correttamente: il dispositivo Serial potrebbe perdere alcuni bit durante lo scambio di controllo fra i task (task switching).

- io_BrkTime. Si deve inizializzare questo parametro con la durata in microsecondi del segnale d'interruzione; il valore di default è 250.000. Il comando rende permanente il valore indicato, anche dopo la chiusura del dispositivo.

- io_TermArray. Si devono memorizzare nelle due long word di questa sotto-struttura i caratteri di EOF "speciali" (si ricordi che questi caratteri devono essere disposti in ordine decrescente). Questo parametro interviene nel trasferimento dei dati solo se risulta impostato il flag SERF_EOFMODE del parametro io_SerFlags. Il comando rende permanenti i caratteri di EOF, anche dopo la chiusura del dispositivo.

- io_ReadLen. Si deve inizializzare questo parametro con il numero di bit (1-8) contenuti in ogni byte ricevuto tramite il comando CMD_READ. Non è incluso il bit di parità. Il comando rende permanente il valore impostato, anche dopo la chiusura del dispositivo.

- io_WriteLen. Si deve inizializzare questo parametro con il numero di bit (1-8) contenuti in ogni byte inviato tramite il comando CMD_WRITE. Non è incluso il bit di parità. Il comando rende permanente il valore impostato, anche dopo la chiusura del dispositivo.

- io_StopBits. Si deve inizializzare questo parametro con il numero di bit di stop (0, 1 oppure 2) da usare nella trasmissione. Due bit di stop possono essere utilizzati per il comando CMD_READ solo se il parametro io_ReadLen vale 7. Il comando rende permanente il valore impostato, anche dopo la chiusura del dispositivo.

- io_SerFlags. Si devono impostare i flag di questo parametro a seconda delle proprie esigenze. Questi flag vengono sempre opportunamente inizializzati durante ogni nuova esecuzione della funzione OpenDevice; essi riflettono lo stato delle routine interne del dispositivo Serial e della sua configurazione.

Discussione

SDCMD_SETPARAMS permette a un task di cambiare direttamente i parametri del dispositivo Serial utilizzati dai comandi CMD_READ e CMD_WRITE. Viene utilizzato per aggiornare i parametri del dispositivo Serial prima di un nuovo comando CMD_READ o CMD_WRITE.

SDCMD_SETPARAMS è particolarmente importante per la definizione dei caratteri di EOF che concludono l'esecuzione di una richiesta di I/O CMD_READ. Esso permette a un task di cambiare la definizione di questi caratteri prima d'inviare il successivo comando CMD_READ o CMD_WRITE. Se un task dialoga con un certo numero di dispositivi hardware esterni che presentano diverse caratteristiche (per esempio, diversi caratteri per la separazione dei blocchi di dati), esso può cambiare l'array di caratteri di EOF in modo che il successivo comando CMD_READ o CMD_WRITE sia in grado di comunicare correttamente con il dispositivo hardware esterno collegato alla porta seriale.

È anche possibile definire un task separato per ogni dispositivo collegato alla porta seriale. Per ogni task vengono definiti caratteri di EOF diversi, in modo da lavorare con le specifiche caratteristiche dell'hardware esterno.

SERF_XDISABLED è l'unico flag che può essere alterato mentre le routine interne del dispositivo Serial stanno eseguendo un comando CMD_READ o CMD_WRITE. Se il task cambia SERF_XDISABLED, il parametro io_Error della struttura IOExtSer utilizzata per CMD_READ o CMD_WRITE viene inizializzato a SerErr_DevBusy.

Si tenga presente che i flag SERF_EOFMODE e SERF_QUEUEDBRK del parametro io_SerFlags possono essere impostati o azzerati direttamente, ovvero senza utilizzare il comando SDCMD_SETPARAMS. Il flag SERF_SHARED può essere impostato prima di chiamare la funzione OpenDevice, mentre non può essere azzerato a dispositivo già aperto. Tutti gli altri parametri del dispositivo Serial possono essere alterati soltanto inviando un comando SDCMD_SETPARAMS.

Se un task intende comunicare con un'interfaccia MIDI (Musical Instrument Digital Interface), deve impostare il flag SERF_RAD_BOOGIE, il quale imposta automaticamente il bit SERF_XDISABLED per eliminare dati non richiesti e attivare la massima velocità per il trasferimento dati. SERF_RAD_BOOGIE evita la verifica della parità, il riconoscimento dei caratteri XON e lunghezze dei byte diverse da otto bit. Esso verifica però la presenza di eventuali segnali d'interruzione.

Il trasferimento di dati nel formato MIDI ad alta velocità è facilmente realizzabile. Tuttavia, l'uso delle sole routine del dispositivo Serial può rivelarsi poco affidabile in quanto il protocollo MIDI richiede una precisa sincronizzazione, e in un sistema multitasking come l'Amiga le routine interne del dispositivo Serial potrebbe perdere vitali cicli della CPU (sottratti dal sistema per svolgere altri compiti), provocando la perdita di caratteri durante la trasmissione.

Selezionando la parità mark o space si provoca l'inizializzazione del flag SERF_PARTY_ON (verfica della parità abilitata) mentre viene ignorato il flag SERF_PARTY_ODD (parità dispari).

Il dispositivo Input

Introduzione

Il dispositivo Input riceve gli eventi di input provenienti da vari dispositivi – Keyboard, Gameport, Timer e TrackDisk – insieme agli eventi di input definiti dai task. Provvede a riunirli in un appropriato flusso concatenato, chiamato *flusso degli eventi di input*, il quale viene poi reso disponibile alle funzioni di gestione degli input (input-handler) definite dal sistema e dal programmatore, che provvedono a elaborare il flusso degli eventi.

Il dispositivo Input risiede su ROM. Nell'Amiga 1000 viene caricato da disco nella ROM WCS (Write Control Store) al momento dell'installazione del Kickstart. Viene aperto indirettamente quando il task apre il dispositivo Console. All'apertura del dispositivo Input, il sistema crea automaticamente un task (task di input), il quale comunica direttamente con i dispositivi per ricevere eventi di input allo stato grezzo, come quelli relativi alla pressione dei tasti sul mouse e al suo movimento, gli eventi di temporizzazione, gli eventi relativi all'inserimento e alla rimozione di un disco da un disk drive e così via (il task di input appare nella lista di sistema TaskWait con il nome "input.device" e con priorità 20). Il task di input può anche ricevere eventi definiti dai task, e questo consente a un task di simulare per esempio i movimenti del mouse, così da ingannare Intuition facendogli credere che il mouse si muove quando in realtà l'utente non lo tocca.

In generale, esistono tre modi per trattare con gli eventi di input. Il primo è quello d'interagire direttamente con i dispositivi che li creano, cioè utilizzare i comandi e le funzioni previsti dal singolo dispositivo per elaborare gli eventi di input che genera. Questo è un metodo che andrebbe evitato.

Il secondo metodo consiste nell'utilizzare una serie di funzioni create dal programmatore per la gestione degli input, ovvero il metodo che discuteremo in questa sede.

Il terzo metodo per ricevere eventi di input è quello di ottenerli dal dispositivo Console oppure da Intuition. In quest'ultimo caso, il task ottiene gli eventi di input dall'IDCMP (Intuition's Direct Communications Message Ports, message port di Intuition per le comunicazioni dirette). Se viene utilizzato questo metodo, occorre prestare attenzione a quello che accade agli eventi di input qualora il task non li prenda in considerazione. Infatti, se non esiste una finestra o un'unità del dispositivo Console attiva, gli eventi di input (pressione dei tasti sulla tastiera o pressione del pulsante sinistro del mouse) vengono ignorati da Intuition. Se invece il task ha aperto una finestra di Intuition, l'ha resa attiva, e lascia accodare gli eventi di input senza rispondervi, si verificano le seguenti situazioni:

- se il sistema non possiede alcuna struttura InputEvent vuota da utilizzare per rappresentare il nuovo evento di input, provvede ad allocarne una e ad accodarla alla message port del task.

- Quando infine il task risponde agli eventi di input, la memoria allocata

per le strutture InputEvent non viene liberata fino a quando la finestra non viene chiusa. Un task che non risponde agli eventi di input per un lungo periodo di tempo può provocare l'allocazione di una grande quantità di memoria. È quindi consigliabile programmare i propri task in modo che rispondano agli eventi di input il più rapidamente possibile, minimizzando l'occupazione della memoria.

Funzionamento del dispositivo Input

La Figura 7.1 (nella pagina successiva) mostra le operazioni svolte dal dispositivo Input. Come viene mostrato, il dispositivo Input riunisce gli eventi di input provenienti da cinque fonti.

- Il dispositivo Keyboard, che è sempre aperto e attivo, invia al dispositivo Input un flusso di eventi da tastiera. Ogni volta che un tasto viene premuto, il dispositivo Keyboard genera un nuovo evento per il dispositivo Input.

- Il dispositivo TrackDisk, che è sempre aperto e attivo, invia al dispositivo Input, attraverso l'AmigaDOS, un evento di input ogni volta che l'utente inserisce o rimuove un disco da un disk drive.

- Il dispositivo Timer, che è sempre aperto e attivo, invia al dispositivo Input un flusso di eventi; questi eventi, rappresentati da singole strutture timeval, caratterizzano le temporizzazioni del sistema.

- Il dispositivo Gameport, se aperto e attivo, invia al dispositivo Input un flusso di eventi provenienti dalle porte giochi. Ogni volta che il mouse (oppure un altro dispositivo hardware di input) cambia stato, viene generato un nuovo evento di input.

- Ogni task può inviare al dispositivo Input strutture di tipo InputEvent che descrivono eventi di input; in questo modo un programmatore può per esempio simulare eventi generati da un particolare dispositivo hardware senza che questo venga realmente impiegato.

Il dispositivo Input riunisce gli eventi di input provenienti da queste cinque fonti in una lista di strutture a concatenazione semplice. Per rappresentare un evento di input viene impiegata la struttura InputEvent, la quale descrive l'evento e riporta nella sotto-struttura timeval l'istante in cui si è verificato. Ogni nuovo evento viene aggiunto alla fine della lista. Il parametro ie_NextEvent della struttura InputEvent serve a concatenare gli eventi fra loro in modo da formare un flusso.

Le routine interne del dispositivo Input concatenano e organizzano tutti gli eventi automaticamente (ad eccezione di alcuni particolari eventi di input definiti dai task, come vedremo nella prossima sezione). Il flusso degli eventi

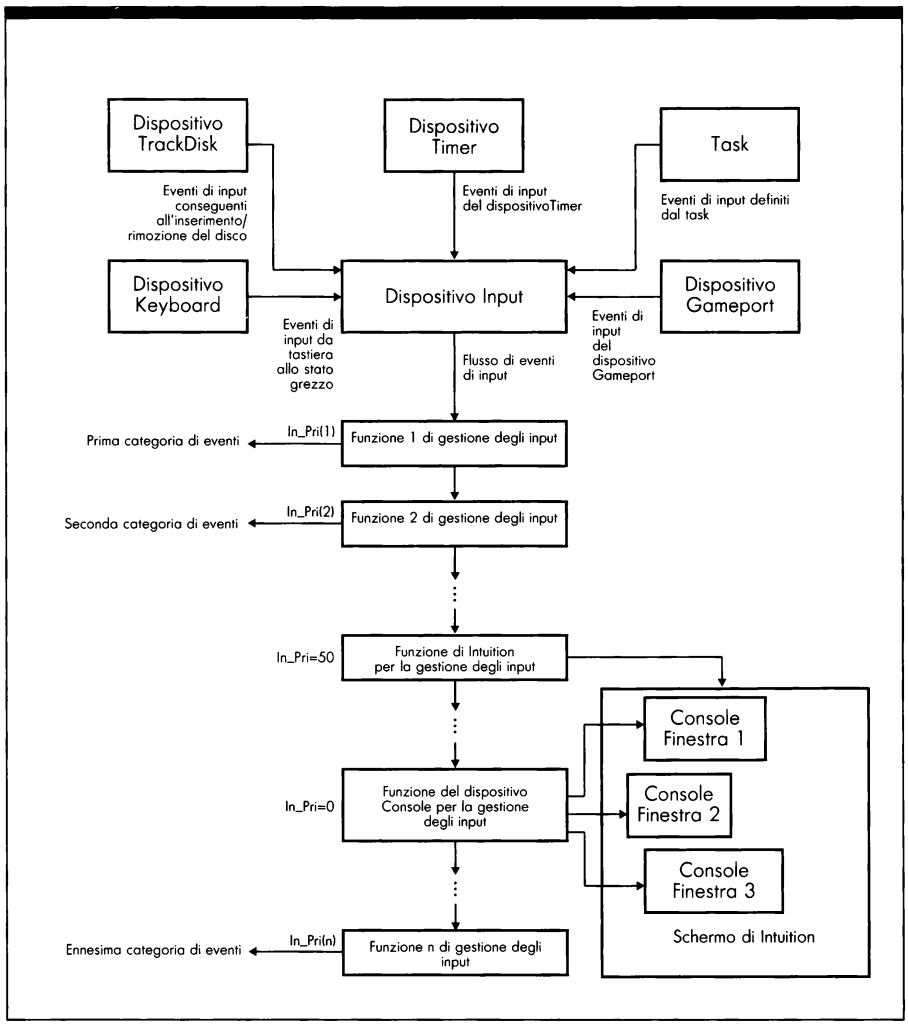


Figura 7.1: *Funzionamento del dispositivo Input*

così ottenuto viene quindi ceduto a una serie di funzioni di gestione, che provvedono a elaborare le proprie categorie di eventi. La sequenza in cui queste funzioni vengono eseguite è determinata dalle loro priorità.

Impiego delle funzioni di gestione degli eventi di input

Le funzioni di gestione degli eventi di input sono funzioni di interrupt alle quali il dispositivo Input cede il flusso perché sia elaborato. Un task può creare proprie funzioni di gestione degli eventi di input per filtrare particolari eventi.

Per esempio, una funzione potrebbe essere progettata per intercettare tutti gli eventi di input provenienti dalla tastiera e cambiarli in altri eventi di input sempre da tastiera (per esempio, allo scopo di cambiare le A in B oppure le B in A); in questo caso gli eventi di input da tastiera verrebbero alterati. Comunque, più in generale, le funzioni di gestione degli eventi di input possono anche eliminare o aggiungere nuovi eventi al flusso.

Un secondo esempio potrebbe essere una funzione di gestione degli eventi di input creata per preelaborare gli eventi di input provenienti da una porta giochi a cui sia collegato un dispositivo hardware non previsto dall'Amiga. Ogni funzione per la gestione degli eventi di input dev'essere progettata per elaborare uno specifico sottoinsieme di eventi.

Le funzioni di gestione degli eventi di input vengono aggiunte al sistema con il comando IND_ADDHANDLER. Una funzione di questo tipo viene progettata specificandone tutti i parametri (come il codice eseguibile e i dati relativi) all'interno di una struttura Interrupt. In questo modo, ogni funzione aggiunta alla lista delle funzioni di gestione degli eventi di input svolge la preelaborazione di un sottoinsieme specifico di eventi di input fino a quando non viene rimossa dal comando IND_REMHANDLER. A ogni funzione presente nella lista viene ceduto l'indirizzo del flusso di eventi di input restituito dalla funzione precedente; questo indirizzo individua la prima struttura InputEvent del flusso. La funzione che detiene il flusso esamina gli eventi alla ricerca di quelli alla cui elaborazione è stata preposta, ignorando invece quelli che non la interessano. Ogni volta che la funzione incontra un evento di input di sua competenza, può modificarlo, oppure eliminarlo dal flusso, così da non farlo pervenire alle altre funzioni di gestione degli eventi di input. Essa può anche creare nuovi eventi di input per Intuition o per altri programmi che dispongono di funzioni di gestione degli eventi di input a priorità inferiore.

A ogni funzione aggiunta al sistema con il comando IND_ADDHANDLER dev'essere assegnata una priorità, che i task devono indicare nel parametro ln_Pri della sotto-struttura Node contenuta nella struttura Interrupt. Indicando la priorità più bassa (−128), la funzione viene inserita alla fine della lista, ed è quindi l'ultima a ricevere il controllo e l'indirizzo del flusso di eventi; in una posizione così sfavorevole la funzione può anche non ricevere mai il controllo se le funzioni a più alta priorità che la precedono estinguono completamente gli eventi di input presenti nel flusso. Se invece viene indicata la più alta priorità (+127), la funzione viene inserita all'inizio della lista, ed è quindi la prima a ricevere il controllo e l'indirizzo del flusso di eventi; in questo caso, la funzione riceve il flusso di eventi nella sua integrità, ed ha quindi la certezza che niente è stato aggiunto, modificato o alterato da altre funzioni. Riepilogando, il flusso degli eventi di input organizzato dal dispositivo Input viene ceduto alla funzione di gestione a più alta priorità. La funzione può compiere qualsiasi operazione sul flusso: può modificarne alcuni eventi, può eliminarli, aggiungerne di nuovi; quando la funzione ha terminato l'elaborazione degli eventi che la interessano, restituisce l'indirizzo della prima struttura InputEvent del nuovo flusso che si è venuto a creare. Questo è l'indirizzo che viene ceduto alla funzione di gestione dotata della priorità appena inferiore. Questo iter prosegue fino a quando il controllo non viene ceduto alla funzione di gestione dotata della più bassa priorità. Gli eventi di input che

eventualmente risultassero ancora nel flusso vengono persi (si ricordi che anche se non appartengono più al flusso, la memoria occupata dalle loro strutture InputEvent non viene liberata).

La funzione di gestione degli eventi di input che il software sistema crea per Intuition viene inserita nella lista con priorità 50. Se si desidera intercettare e filtrare gli eventi di input prima che il flusso venga elaborato da Intuition, occorre aggiungere alla lista la propria funzione per la gestione degli eventi di input indicando una priorità maggiore di 50.

La preelaborazione è un procedimento continuo; gli eventi di input vengono inviati al sistema dalle sorgenti di input, mentre contemporaneamente l'ultimo flusso creato dal dispositivo Input viene elaborato dalle funzioni concatenate nella lista. Parallelamente i task possono aggiungere e rimuovere funzioni dalla lista, in modo da poter elaborare correttamente gli eventi e svolgere le proprie mansioni. Questo procedimento dinamico in continua evoluzione viene per la maggior parte del tempo controllato delle routine di sistema e dal dispositivo Input.

Nella creazione delle funzioni di gestione degli eventi di input occorre sempre osservare le seguenti regole:

- se la funzione, quando riceve il controllo e l'indirizzo del flusso di eventi, rileva che nel flusso è presente una sola struttura InputEvent (il relativo parametro ie_NextEvent contiene un indirizzo nullo), e tale struttura corrisponde a un evento di input che essa deve elaborare ed eliminare, in pratica causa l'estinzione del flusso. Deve quindi restituire un indirizzo nullo, per indicare che il flusso di eventi è vuoto. Le funzioni di gestione degli eventi che seguono con priorità inferiori non riceveranno il controllo.

- Se la funzione, quando riceve il controllo e l'indirizzo del flusso di eventi, rileva che nel flusso sono presenti più strutture InputEvent tra loro concatenate, dev'essere in grado di scorrere la lista alla ricerca degli eventi che la interessano. L'elaborazione del flusso può comportare operazioni come la modifica, l'eliminazione e l'aggiunta di eventi (che vedremo fra poco), cioè l'alterazione del flusso. Durante queste operazioni, la funzione deve preoccuparsi di mantenere il concatenamento fra le strutture InputEvent che costituiscono il flusso. Terminata l'elaborazione del flusso, la funzione deve restituire l'indirizzo della struttura InputEvent che definisce il primo evento di input nel nuovo flusso che si è venuto a creare. La funzione con la priorità appena inferiore riceverà l'indirizzo di questo nuovo flusso.

- Se la funzione desidera aggiungere nuovi eventi per cederli successivamente alle funzioni con priorità più bassa, deve allocare in RAM le strutture InputEvent necessarie per definirli e aggiungerle al flusso. Quando questa funzione riottiene il controllo dopo un ciclo completo di gestione degli eventi, non deve fare nessuna supposizione riguardo al contenuto delle strutture InputEvent che aveva aggiunto durante il ciclo precedente, dal momento che potrebbero essere state modificate da

altre funzioni. La funzione che aggiunge strutture InputEvent deve anche mantenere traccia dell'indirizzo di partenza e dell'ammontare di memoria RAM utilizzata dalle strutture InputEvent che ha aggiunto, in modo da poterla liberare quando riottiene il controllo per elaborare un nuovo flusso.

comandi del dispositivo Input

Per interagire con il dispositivo Input si utilizzano otto comandi specifici, e solo quattro comandi standard. Di ognuno di questi comandi indichiamo esplicitamente la natura: se è di tipo immediato, se permette il QuickIO oppure solo il QueuedIO. Tutti i comandi del dispositivo Input influenzano il parametro io_Error della struttura IOStdReq oppure della struttura timerequest.

L'invio dei comandi al dispositivo Input

La Figura 7.2 (nella pagina successiva) mostra lo schema generale utilizzato per inviare comandi al dispositivo Input. Le linee con le frecce rappresentano i parametri che devono essere inizializzati e quelli restituiti dalle routine interne del dispositivo.

L'impiego del dispositivo Input prevede tre fasi.

1. *Preparazione delle strutture*. In questa fase si inizializzano i consueti parametri delle strutture IOStdReq, timerequest e Interrupt per poter poi inviare un comando alle routine interne del dispositivo Input. Inoltre, occorre inizializzare i puntatori is_Data e is_Code della struttura Interrupt per l'invio dei comandi IND_ADDHANDLER e IND_REMHANDLER, e i parametri tv_secs e tv_micro per l'inoltro dei comandi IND_SETPERIOD e IND_SETTHRESH. La scelta dei parametri da inizializzare dipende dallo specifico comando.

2. *Invio del comando e sua elaborazione*. L'unico compito che il programmatore svolge in questa fase è quello d'inviare il comando al dispositivo utilizzando le funzioni BeginIO, DoIO o SendIO. Il controllo passa alle routine interne del dispositivo e del sistema.

3. *Elaborazione dei parametri di output del comando e loro restituzione*. In questa fase il sistema e il dispositivo Input possiedono il completo controllo. Qui i risultati prodotti dall'elaborazione della richiesta vengono restituiti al task che l'aveva inviata. Se il comando non è immediato e la richiesta non è stata inviata richiedendo il QuickIO, viene elaborata dal dispositivo nel momento in cui raggiunge la sommità della coda alla request port e successivamente viene inserita nella coda alla reply port del task. Nel caso invece che sia stato richiesto il QuickIO, oppure nel caso che il comando sia uno di quelli che il

dispositivo esegue in modo immediato, non si verifica alcun accodamento alla request port e la richiesta giunge direttamente alle routine interne del dispositivo.

La Figura 7.2 illustra, inoltre, i parametri significativi nell'inizializzazione e nell'elaborazione delle funzioni del dispositivo Input. La funzione OpenDevice aggiorna i parametri io_Device e io_Unit rispettivamente con gli indirizzi della struttura Device relativa al dispositivo e della struttura Unit relativa all'unica unità posseduta dal dispositivo. La funzione CloseDevice, invece, azzera i puntatori io_Device e io_Unit della struttura di I/O e decrementa di 1 il parametro lib_OpenCnt della struttura Device. Infine, OpenDevice influenza anche il parametro io_Error, nel quale memorizza l'esito della sua esecuzione.

Le strutture del dispositivo Input

La Figura 7.3 (a pagina 245) illustra le strutture necessarie per interagire con il dispositivo Input. Quando non sta comunicando con un task, esso tratta direttamente solo con la struttura InputEvent, quella che definisce gli eventi di input provenienti dai dispositivi Gameport, Console, TrackDisk, Timer e dai task. I task per interagire con le sue routine interne, cioè per inviare comandi, devono impiegare le strutture IOStdReq, timerequest e Interrupt, utilizzate per definire le informazioni richieste dal dispositivo Input.

La struttura InputEvent definisce le caratteristiche dell'evento di input. Fra i suoi parametri compaiono un'unione e una sotto-struttura. L'unione ie_Position, a seconda del tipo di evento, può costituire una struttura denominata ie_xy, adibita a riportare la posizione del mouse, o un puntatore di tipo APTR denominato ie_addr, adibito a indicare in memoria i dati che

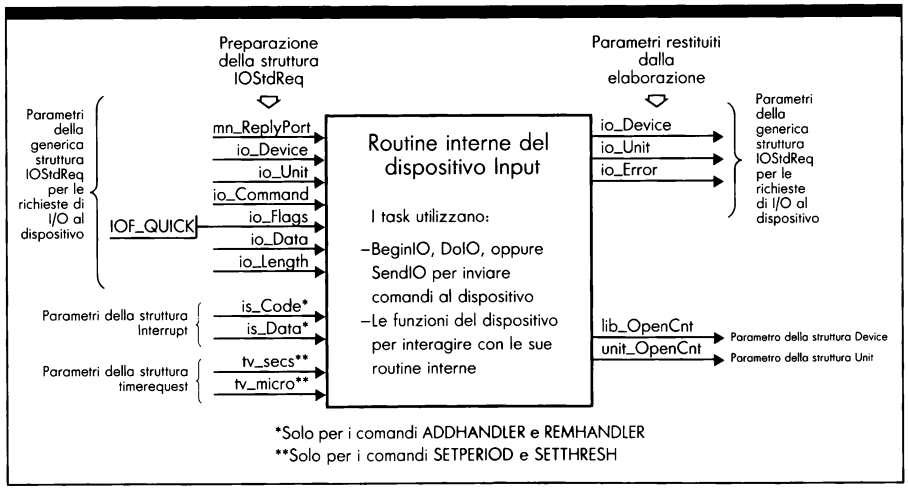


Figura 7.2: *Gestione delle funzioni e dei comandi previsti dal dispositivo Input*

descrivono l'evento (ovviamente, la struttura ie_xy e il puntatore ie_addr occupano entrambi lo stesso numero di byte); l'impiego dell'unione permette di risparmiare memoria, ed è giustificato dal fatto che per il singolo evento di input le due informazioni (posizione del mouse e indirizzo dei dati che descrivono l'evento) si escludono a vicenda. La sotto-struttura ie_TimeStamp, di tipo timeval, viene utilizzata per registrare il momento in cui l'evento di input si è verificato. Nella struttura InputEvent compare anche il puntatore ie_NextEvent, che serve a individuare in memoria la successiva struttura InputEvent del flusso.

La struttura Interrupt viene normalmente impiegata per definire la routine di gestione di un interrupt software. Il task deve impiegarla con il dispositivo Input quando desidera aggiungere (IND_ADDHANDLER) o rimuovere (IND_REMHANDLER) una funzione di gestione degli eventi di input: deve indicare nel parametro is_Code il punto d'ingresso della funzione e nel parametro is_Data la posizione in memoria della relativa area dati.

Quando questa funzione termina l'elaborazione del flusso di eventi, deve restituire l'indirizzo del nuovo flusso che si è venuto a creare. Questo indirizzo viene ceduto dal dispositivo Input alla successiva funzione di gestione degli eventi di input, quella a priorità appena inferiore.

La struttura IOStdReq viene utilizzata per definire e inviare le richieste di I/O al dispositivo. Costituisce un messaggio nel quale, al pari degli altri dispositivi, il task indica un comando da eseguire e i relativi parametri.

La struttura timerequest consiste di una sotto-struttura IORequest e una sotto-struttura timeval. Viene utilizzata come struttura per le richieste di I/O relative ai comandi IND_SETPERIOD e IND_SETTHRESH; questi due comandi permettono d'impostare alcune caratteristiche della tastiera dell'Amiga. Per una trattazione più completa dell'argomento si veda il capitolo 13.

La struttura InputEvent

La struttura InputEvent è definita come segue:

```
struct InputEvent {
        struct InputEvent *ie_NextEvent;
        UBYTE ie_Class;
        UBYTE ie_SubClass;
        UWORD ie_Code;
        UWORD ie_Qualifier;
        union {
                struct {
                        WORD ie_x;
                        WORD ie_y;
                } ie_xy;
                APTR ie_addr;
        } ie_position;
        struct timeval ie_TimeStamp;
};
```

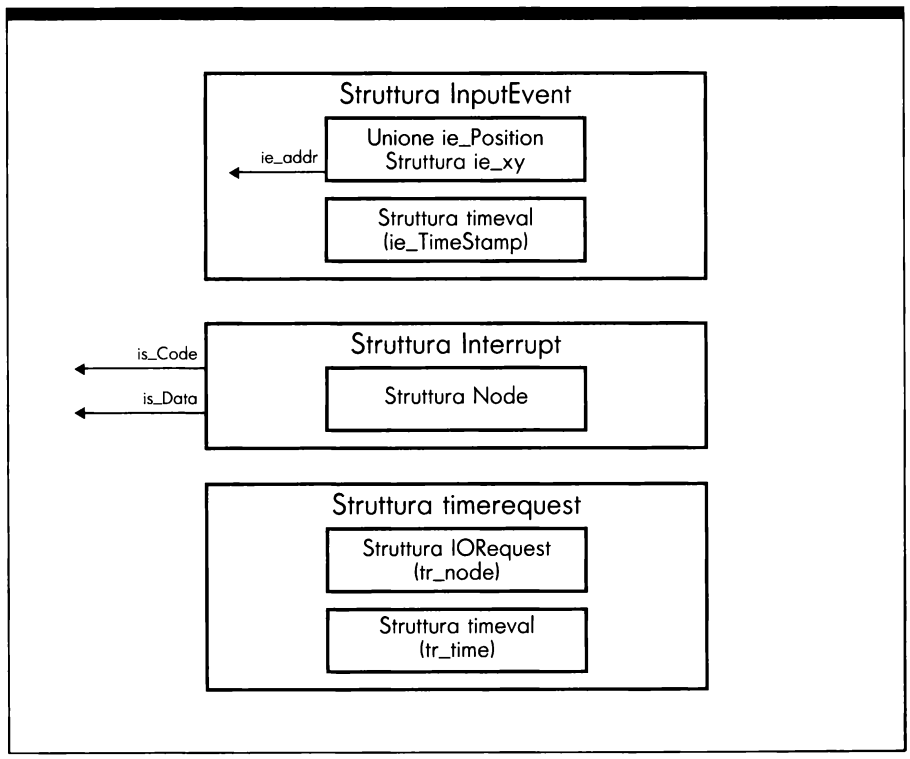


Figura 7.3: *Strutture utilizzate dal dispositivo Input*

I parametri della struttura InputEvent hanno il seguente significato:

- ie_NextEvent contiene l'indirizzo della struttura InputEvent che rappresenta il successivo evento nella lista concatenata. A prescindere dalla sua origine, questo è l'evento successivo anche cronologicamente.

- ie_Class contiene un codice che rappresenta la classe dell'evento di input. Generalmente permette al task d'individuarne la natura. Per esempio la selezione di un gadget, la selezione del gadget di chiusura di una finestra, l'inserimento di un disco in un disk drive...

- ie_SubClass può contenere un codice che rappresenta la sotto-classe dell'evento di input (opzionale). Si tratta di un parametro che può essere impiegato dai dispositivi per descrivere meglio l'evento di input.

- ie_Code è il codice dell'evento di input. Per gli eventi da tastiera si tratta del codice del tasto premuto; se l'evento proviene dalla porta giochi riporta informazioni sullo stato dei pulsanti.

- ie_Qualifier contiene il "qualificatore" dell'evento, un codice che per

esempio potrebbe significare "Tasto Alt contemporaneamente premuto". I qualificatori sono discussi più avanti.

- ie_x è la coordinata X della posizione del puntatore quando si è verificato l'evento di input.
- ie_y è la coordinata Y della posizione del puntatore quando si è verificato l'evento di input.
- ie_xy è il nome della sotto-struttura contenuta nell'unione ie_position che riporta la posizione del mouse.
- ie_addr è l'indirizzo dell'evento di input. Questa informazione è il secondo parametro dell'unione ie_position, e come tale occupa gli stessi quattro byte occupati dalla sotto-struttura ie_xy. Quindi, le due informazioni si escludono a vicenda.
- ie_position è il nome dell'unione indicata nella struttura InputEvent.
- ie_TimeStamp è il nome della sotto-struttura di tipo timeval presente all'interno della struttura InputEvent; questa sotto-struttura indica il momento in cui si è verificato l'evento.

I qualificatori attualmente previsti sono i seguenti:

- IEQUALIFIER_LSHIFT. Indica che è stato premuto il tasto Shift di sinistra.
- IEQUALIFIER_RSHIFT. Indica che è stato premuto il tasto Shift di destra.
- IEQUALIFIER_CAPSLOCK. Indica che è stato premuto il tasto che abilita le maiuscole (Capitals Lock).
- IEQUALIFIER_CONTROL. Indica che è stato premuto il tasto Ctrl.
- IEQUALIFIER_LALT. Indica che è stato premuto il tasto Alt di sinistra.
- IEQUALIFIER_RALT. Indica che è stato premuto il tasto Alt di destra.
- IEQUALIFIER_LCOMMAND. Indica che è stato premuto il tasto Amiga di sinistra.
- IEQUALIFIER_RCOMMAND. Indica che è stato premuto il tasto Amiga di destra.
- IEQUALIFIER_NUMERICPAD. Indica che è stato premuto un tasto della tastierina numerica.

- IEQUALIFIER_REPEAT. Indica che si è ripetuto il precedente evento di input da tastiera.
- IEQUALIFIER_INTERRUPT. Indica che si è verificato un interrupt di sistema.
- IEQUALIFIER_MULTIBROADCAST. L'evento verrà inviato a tutte le finestre di Intuition aperte e non solo alla finestra attiva.
- IEQUALIFIER_LBUTTON. Indica che è stato premuto il tasto di sinistra del mouse.
- IEQUALIFIER_RBUTTON. Indica che è stato premuto il tasto di destra del mouse.
- IEQUALIFIER_MBUTTON. Indica che è stato premuto il tasto centrale del mouse (questo tasto non esiste nel mouse standard fornito con l'Amiga).
- IEQUALIFIER_RELATIVEMOUSE. Le coordinate fornite dal mouse sono relative a una particolare posizione, e quindi non sono assolute.

Per ottenere maggiori informazioni sulle classi degli eventi di input, sui codici, sui qualificatori, si consulti il file INCLUDE denominato inputevent.h.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOStdReq)
A1

Scopo della funzione

Questa funzione chiude l'accesso da parte del task all'unica unità del dispositivo Input, l'unità 0. Se nessun altro task sta utilizzando il dispositivo Input e il dispositivo Console risulta chiuso, anche i dispositivi Timer, Keyboard

e Gameport vengono automaticamente chiusi con l'esecuzione della funzione. Inoltre, CloseDevice decrementa il parametro lib_OpenCnt contenuto nella struttura Device relativa al dispositivo.

La funzione CloseDevice provvede inoltre ad aggiornare con il valore 0 i puntatori io_Device e io_Unit della struttura IOStdReq. Dopo aver eseguito la funzione CloseDevice, per accedere al dispositivo il task deve chiamare ancora OpenDevice.

Argomenti della funzione

iOStdReq Dev'essere l'indirizzo della struttura di tipo IOStdReq che il task impiega per interagire con il dispositivo Input. Generalmente questa struttura è la stessa che viene parzialmente inizializzata dalla funzione OpenDevice quando il task apre il dispositivo. Si noti che comunque questo indirizzo può anche indicare una diversa struttura da quella utilizzata con OpenDevice, a patto però che i parametri io_Device e io_Unit siano quelli che OpenDevice ha restituito.

Discussione

CloseDevice permette a un task di concludere l'accesso al dispositivo Input. Sebbene questo dispositivo permetta esclusivamente l'accesso condiviso, è sempre buona norma che il task lo chiuda quando non ne ha più bisogno. Prima di chiamare CloseDevice deve però verificare che tutte le richieste di I/O inviate abbiano ricevuto la relativa risposta dalle routine interne del dispositivo Serial. È possibile effettuare questa operazione utilizzando le funzioni GetMsg, CheckIO e WaitIO, le quali verificano, in modi diversi, la presenza di richieste nella coda alla reply port del task.

CloseDevice chiude automaticamente anche i dispositivi Timer, Keyboard e Gameport per il task che l'ha chiamata. Tuttavia, dato che tutti questi dispositivi operano in accesso condiviso, essi possono rimanere attivi per altri task che li hanno aperti esplicitamente con la funzione OpenDevice oppure indirettamente tramite i dispositivi Input o Console.

Il task di input del sistema apre il dispositivo Input come parte della sequenza di inizializzazione del sistema. Se il sistema chiude il dispositivo Input, l'elaborazione degli eventi di input non può avvenire; in questo caso la macchina non risponde a nessun evento proveniente dalla tastiera o dalle porte giochi.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("input.device", ØL, iOStdReq, ØL)
DØ                  AØ              DØ A1        D1
```

Scopo della funzione

Questa funzione apre l'accesso da parte del task all'unità 0, l'unica prevista dal dispositivo Input. Con l'esecuzione di OpenDevice vengono aperti anche i dispositivi Timer, Gameport e Keyboard. Il dispositivo Input opera sempre nel modo di accesso condiviso.

OpenDevice incrementa di 1 il parametro lib_OpenCnt appartenente alla struttura Device di gestione relativa al dispositivo. I risultati prodotti dall'esecuzione della funzione sono i seguenti:

- io_Device. La funzione memorizza in questo puntatore l'indirizzo della struttura Device di gestione del dispositivo Input. Si tratta di una struttura di tipo Library che definisce la libreria del dispositivo.
- io_Unit. La funzione memorizza in questo puntatore l'indirizzo della struttura Unit di gestione dell'unità. Si noti che il parametro unit_OpenCnt di questa struttura non viene impiegato per tenere il conto di quante volte l'unità viene aperta. La struttura Unit contiene anche una struttura MsgPort che rappresenta la coda alla request port.
- io_Error. Il valore 0 indica che la richiesta di apertura del dispositivo ha avuto successo. IOERR_OPENFAIL indica che il dispositivo Input non è stato aperto (in genere per una carenza di memoria).

Argomenti della funzione

"input.device" Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Input.

ØL Questo argomento indica che si desidera aprire l'unità 0, l'unica disponibile nel dispositivo Input.

iOStdReq Deve contenere l'indirizzo della struttura di tipo

IOStdReq che il task impiega per interagire con il dispositivo.

ØL Indica che la funzione ignora l'argomento flag.

Preparazione della struttura IOStdReq

Per aprire il dispositivo Input, occorre inizializzare il parametro mn_ReplyPort della struttura di I/O con l'indirizzo della struttura MsgPort che rappresenta la reply port del task. Il task può allocare una message port tramite la funzione CreatePort di supporto alla libreria Exec, e indicarne l'indirizzo come argomento della funzione CreateExtIO. Chiamando quest'ultima funzione il task alloca la struttura di I/O necessaria per interagire con il dispositivo e memorizza automaticamente l'indirizzo della reply port nel parametro mn_ReplyPort.

Discussione

OpenDevice permette a un task di accedere al dispositivo Input. Oltre che inizializzare gli appropriati parametri, il task può fare in modo che con la chiamata venga automaticamente inoltrato un comando IND_WRITEEVENT. Una volta che il task ha ottenuto l'accesso al dispositivo Input, può inviare una serie di comandi IND_WRITEEVENT (tramite le funzioni BeginIO, DoIO oppure SendIO) per definire i propri eventi di input.

COMANDI STANDARD DEL DISPOSITIVO

CMD_FLUSH

Scopo del comando

Il comando CMD_FLUSH elimina tutte le richieste di I/O accodate alla request port del dispositivo, mentre i comandi in esecuzione non vengono influenzati. Tutte le richieste di I/O eliminate vengono restituite alla reply port del task con il codice d'errore IOERR_ABORTED nel parametro io_Error.

CMD_FLUSH viene sempre eseguito in modo immediato. L'esito del

comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo; IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_FLUSH e azzerare il parametro io_Flags.

Discussione

Il comando CMD_FLUSH elimina tutte le richieste di I/O presenti nella coda alla request port dell'unità. Dato che CMD_FLUSH è un comando che ha effetti distruttivi, si dovrebbe utilizzarlo solo per riportare il dispositivo alle condizioni iniziali, con la coda alla request port completamente vuota.

CMD_RESET

Scopo del comando

CMD_RESET ripristina tutti i valori di default dei parametri e dei flag del dispositivo Input. Elimina tutte le richieste di I/O, attive o accodate, e chiama CMD_START per riattivare l'unità nel caso che sia stata precedentemente bloccata con il comando CMD_STOP. Tutte le richieste di I/O eliminate vengono restituite alla reply port del task con il codice d'errore IOERR_ABORTED nel parametro io_Error.

CMD_RESET viene sempre eseguito in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_RESET e azzerare il parametro io_Flags.

Discussione

CMD_RESET è un comando che ha effetti distruttivi. Esso chiama indirettamente CMD_FLUSH, eliminando tutte le richieste di I/O accodate nella coda dell'unità. Elimina anche tutte le richieste di I/O attive.

CMD_START

Scopo del comando

Se l'unità del dispositivo è stata bloccata con il comando CMD_STOP, CMD_START la riattiva. La riattivazione riguarda qualsiasi operazione interrotta. Inoltre, il comando CMD_START ordina al dispositivo di riprendere la gestione della coda alla request port: le richieste di I/O in essa presenti riprendono la loro ascesa verso la cima della coda. CMD_START azzera sempre il parametro io_Error della struttura IOStdReq e viene sempre eseguito in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice. Il task deve impostare

io_Command con il comando CMD_START e azzerare il parametro io_Flags.

Discussione

CMD_START riattiva le operazioni di scrittura e lettura dei dati bloccate tramite il comando CMD_STOP. CMD_START fa ovviamente riprendere anche l'ascesa dei comandi accodati.

CMD_STOP

Scopo del comando

Il comando CMD_STOP sospende immediatamente le attività dell'unità. Quest'azione evita anche che le routine del dispositivo Input continuino a elaborare le richieste di I/O accodate, con la conseguente crescita della coda a mano a mano che giungono altre richieste di I/O.

CMD_STOP viene sempre eseguito in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_STOP e azzerare il parametro io_Flags.

Discussione

Il comando CMD_STOP blocca alla prima occasione l'elaborazione delle richieste di I/O. Per far riprendere l'attività occorre inviare il comando CMD_START.

COMANDI SPECIFICI DEL DISPOSITIVO

IND_ADDHANDLER

Scopo del comando

Il comando IND_ADDHANDLER aggiunge una nuova funzione tra quelle che elaborano il flusso degli eventi di input.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice.

Il task deve impostare io_Command con il comando IND_ADDHANDLER; azzerare il parametro io_Flags; inizializzare io_Data in modo che punti a una struttura Interrupt che rappresenti la nuova funzione da aggiungere alla lista; configurare la struttura Interrupt in modo che il suo parametro is_Code punti ai codici della funzione (il suo punto d'ingresso), e in modo che il suo parametro is_Data punti all'area allocata per la funzione. L'indirizzo che si memorizza nel parametro is_Data viene poi passato come argomento handlerData alla funzione HandlerFunction (nome generico per le funzioni di gestione degli eventi di input). Infine, il task deve indicare la priorità della funzione nel parametro ln_Pri della struttura Node contenuta nella struttura Interrupt. Questa priorità stabilisce la posizione della funzione all'interno della lista mantenuta dal sistema. Se per esempio il task desidera che la sua funzione di gestione degli eventi di input scavalchi Intuition nell'analisi del flusso di eventi, deve indicare una priorità superiore a 50.

Discussione

IND_ADDHANDLER aggiunge una nuova funzione per la gestione degli eventi di input alla lista di sistema. La funzione che viene aggiunta si deve poter chiamare nel modo seguente:

newInputEvent = HandlerFunction(oldInputEvent, handlerData)

HandlerFunction è il nome della funzione aggiunta dal task e ne costituisce quindi il punto d'ingresso (entry point); nell'argomento oldInputEvent la funzione dev'essere preparata a ricevere l'indirizzo della prima struttura InputEvent del flusso di eventi, indirizzo restituito dalla funzione dotata di priorità immediatamente superiore; nell'argomento handlerData il sistema indica alla funzione l'area di memoria che essa può impiegare per i propri dati (si noti che si tratta dell'indirizzo indicato dal task nel parametro is_Data della struttura Interrupt). Infine, la funzione, dopo aver elaborato l'intero flusso di eventi, deve restituire l'indirizzo della prima struttura InputEvent del nuovo flusso che si è venuto a creare. Questo indirizzo è quello che viene passato nell'argomento oldInputEvent alla successiva funzione nella lista.

Quando una funzione HandlerFunction restituisce un indirizzo nullo. significa che tutti gli eventi di input presenti nel flusso sono stati rimossi.

IND_REMHANDLER

Scopo del comando

Il comando IND_REMHANDLER rimuove una funzione per la gestione degli eventi di input dalla lista di sistema. La funzione rimossa dev'essere una di quelle precedentemente aggiunte alla lista con il comando IND_ADDHANDLER.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit

devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice.

Il task deve impostare io_Command con il comando IND_REMHANDLER; azzerare il parametro io_Flags; inizializzare io_Data in modo che punti alla struttura Interrupt che rappresenta la funzione da rimuovere.

Discussione

IND_REMHANDLER rimuove una funzione di gestione degli eventi di input dalla lista mantenuta dal sistema. Entrambi i comandi IND_ADDHANDLER e IND_REMHANDLER si servono della struttura Interrupt per rappresentare i codici e i dati che definiscono la funzione.

IND_SETMPORT

Scopo del comando

Il comando IND_SETMPORT permette a un task d'indicare al dispositivo Input a quale porta giochi è connesso il mouse: il parametro io_Data della struttura IOStdReq dev'essere inizializzato dal task con l'indirizzo di un apposito byte in RAM. Se il task memorizza in questo byte il valore 0, significa che il mouse è collegato alla porta giochi di sinistra; se invece vi memorizza il valore 1, significa che il mouse è collegato alla porta giochi di destra.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo; IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH. IOERR_BADLENGTH indica che il task ha specificato il parametro io_Length in modo non corretto (il significato di questo parametro verrà illustrato tra poco).

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task

ottiene quando, chiamando la funzione OpenDevice, apre il dispositivo.

Il task deve impostare io_Command con il comando IND_SETMPORT; azzerare il parametro io_Flags o inizializzare il flag IOF_QUICK se desidera richiedere il QuickIO; inizializzare io_Length a 1; inizializzare io_Data in modo che punti a un byte nella RAM; memorizzare in questo byte il valore 0 per indicare che gli eventi di input del mouse provengono dalla porta giochi di sinistra, altrimenti il valore 1 per indicare che gli eventi di input del mouse provengono dalla porta giochi di destra.

Discussione

IND_SETMPORT permette a un task di specificare a quale delle due porte giochi è connesso il mouse. Di solito si tratta della porta giochi 1 (sull'Amiga 2000 è la porta giochi frontale di sinistra; sull'Amiga 500 è la porta giochi di sinistra sul retro della macchina; sull'Amiga 1000 è la porta giochi di sinistra sul lato destro della macchina). Tuttavia, è sempre possibile collegare il mouse alla porta giochi di destra e comunicarlo al sistema. Se il mouse viene spostato, qualsiasi programma che dipenda dal mouse deve controllare gli input del mouse provenienti dalla nuova porta.

I programmi possono richiedere all'utente, per esempio tramite un requester di Intuition, di collegare il mouse a una delle due porte giochi, e nel contempo impartire il comando IND_SETMPORT per abilitare quella porta.

IND_SETMTRIG

Scopo del comando

Il comando IND_SETMTRIG stabilisce quali condizioni devono verificarsi perché il dispositivo Gameport possa rispondere a un comando GPD_READEVENT in attesa.

L'esito del comando IND_SETMTRIG viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH. IOERR_BADLENGTH indica che il task ha specificato il parametro io_Length in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice.

Il task deve impostare io_Command con il comando IND_SETMTRIG; azzerare il parametro io_Flags o inizializzarlo a IOF_QUICK per richiedere il QuickIO; inizializzare io_Length con la dimensione della struttura GameportTrigger; inizializzare io_Data con l'indirizzo della struttura GameportTrigger. Questa struttura è composta dai seguenti parametri:

- gpt_Keys — Impostando il flag GPTF_DOWNKEYS, la pressione del pulsante del mouse genera un evento di input. Impostando il flag GPTF_UPKEYS, il rilascio del pulsante del mouse genera un evento di input.
- gpt_Timeout — Quando viene superato il periodo di tempo impostato in questo parametro (espresso in cinquantesimi di secondo) si ottiene un evento di input.
- gpt_XDelta — Quando viene superata la distanza orizzontale impostata in questo parametro si ottiene un evento di input.
- gpt_YDelta — Quando viene superata la distanza verticale impostata in questo parametro si ottiene un evento di input.

Discussione

IND_SETMTRIG stabilisce le condizioni che devono verificarsi perché venga soddisfatto un comando GPD_READEVENT inviato al dispositivo Gameport. Le condizioni permettono di stabilire se il movimento impresso al mouse ha provocato un cambiamento tale da costituire un evento di input valido. Maggiori dettagli sul comando GPD_READEVENT e sulla struttura GameportTrigger si trovano nel capitolo 10, dedicato al dispositivo Gameport.

IND_SETMTYPE

Scopo del comando

Il comando IND_SETMTYPE permette a un task d'indicare il tipo di dispositivo hardware collegato alla porta giochi, in modo che i segnali da essa provenienti possano essere correttamente interpretati dall'hardware dell'Amiga e dal task di gestione del mouse. A questo scopo, il parametro io_Data della struttura IOStdReq deve puntare a uno speciale byte situato nella memoria RAM.

L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito con successo. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH. IOERR_BADLENGTH indica che il task ha specificato il parametro io_Length in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice.

Il task deve impostare io_Command con il comando IND_SETMTYPE; inizializzare io_Flags a IOF_QUICK se desidera utilizzare il QuickIO, altrimenti inizializzare io_Length con il valore 1; inizializzare io_Data in modo che punti a un byte nella RAM. I significati dei valori che può assumere questo byte sono i seguenti:

- se il byte vale −1, significa che il dispositivo hardware è assegnato a un altro task.
- Se il byte vale 0, significa che il dispositivo hardware non è fra quelli che l'Amiga è in grado di riconoscere.
- Se il byte vale 1, significa che il dispositivo hardware è il normale mouse dell'Amiga.
- Se il byte vale 2, significa che il dispositivo hardware è un joystick relativo.

- Se il byte vale 3, significa che il dispositivo hardware è un joystick assoluto.

Discussione

IND_SETMTYPE indica al dispositivo Input quale tipo di dispositivo hardware è collegato alla porta giochi. Il task può proporre all'utente un ventaglio di dispositivi hardware, e tramite questo comando segnalare al dispositivo Input quale è stato scelto.

I tipi di dispositivi ammessi sono definiti nel file INCLUDE denominato gameport.h. I valori utilizzati per il byte sono GPCT_ALLOCATED, GPCT_NOCONTROLLER, GPCT_MOUSE, GPCT_RELJOYSTICK e GPCT_ABSJOYSTICK, che corrispondono rispettivamente ai valori −1, 0, 1, 2, 3. Si consulti il capitolo 10 per maggiori informazioni su questi valori.

IND_SETPERIOD

Scopo del comando

Il comando IND_SETPERIOD stabilisce l'intervallo di tempo che deve trascorrere fra ogni evento di input. Se si tiene premuto un tasto per un periodo di tempo superiore a quello stabilito con il comando IND_SETTHRESH, nuovi eventi di input vengono generati in sequenza dalla tastiera fino a quando il tasto non viene rilasciato. L'intervallo con cui questi eventi di input addizionali vengono generati e inviati alle routine interne del dispositivo Input è quello impostata dal comando IND_SETPERIOD. Il comando vale per tutti i tasti della tastiera, e rimane immutato fino a quando non viene cambiato con un altro comando IND_SETPERIOD. La struttura a esso relativa è la struttura estesa non standard timerequest.

IND_SETPERIOD viene sempre eseguito in modo immediato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito con successo. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH.

Preparazione della struttura timerequest

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo

MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ha ottenuto chiamando la funzione OpenDevice.

Il task deve impostare io_Command con il comando IND_SETPERIOD; azzerare il parametro io_Flags e inizializzare i seguenti parametri:

- tv_secs. Si deve inizializzare questo parametro con il numero di secondi che costituiscono il periodo di ripetizione dei tasti. Questo parametro può anche essere modificato tramite il programma Preferences del Workbench.

- tv_micro. Si deve inizializzare questo parametro con il numero di microsecondi che vengono sommati al parametro tv_secs per ottenere il periodo di ripetizione dei tasti. Il periodo di tempo è quindi numericamente espresso da una parte intera (in secondi) e da una parte decimale (in microsecondi). Anche questo parametro può essere modificato tramite il programma Preferences del Workbench.

Discussione

Il comando IND_SETPERIOD permette a un task di specificare il tempo che deve trascorrere tra i vari eventi di input inviati alle routine interne del dispositivo Input. Per esempio, se un task desidera ignorare temporaneamente gli eventi di input provenienti dalla tastiera, può inizializzare il parametro tv_secs con valori molto grandi. Così facendo, indipendentemente dal tempo di pressione di un tasto, le routine interne del dispositivo Input non ricevono eventi di input generati dalla tastiera per tutto il periodo indicato. Se invece un task desidera che la ripetizione dei tasti sia molto veloce, può inizializzare a zero il parametro tv_secs e con valori molto piccoli il parametro tv_micro.

I comandi IND_SETPERIOD e IND_SETTHRESH permettono a un task di specificare il modo e la rapidità con cui desidera dialogare con l'utente tramite la tastiera. Di solito questi comandi vengono inviati da Intuition e i loro valori di default sono rappresentati nella sotto-struttura timeval della struttura Preferences di Intuition; possono quindi essere modificati indirettamente tramite il programma Preferences del Workbench. Entrambi vengono inoltrati utilizzando la struttura timerequest.

IND_SETTHRESH

Scopo del comando

Il comando IND_SETTHRESH stabilisce il limite di tempo dopo il quale un tasto premuto inizia a generare nuovi eventi di input (l'intervallo tra questi eventi è invece impostato dal comando IND_SETPERIOD). Questo valore rimane immutato fino a quando non viene eseguito un altro comando IND_SETTHRESH e vale per tutti i tasti della tastiera. IND_SETTHRESH viene sempre eseguito in modo immediato e l'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito con successo. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH.

Preparazione della struttura timerequest

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice. Il task deve impostare io_Command con il comando IND_SETTHRESH e azzerare il parametro io_Flags.

Si devono inoltre inizializzare i seguenti parametri specifici del comando:

- tv_secs. Si deve inizializzare questo parametro con il numero di secondi di attesa prima che un tasto premuto inizi a generare nuovi eventi di input. Questo parametro può essere modificato anche tramite il programma Preferences del Workbench.

- tv_micro. Si deve inizializzare questo parametro con il numero di microsecondi che vengono sommati al parametro tv_Secs per ottenere il tempo di attesa prima che un tasto premuto inizi a generare nuovi eventi di input. Il periodo di tempo è quindi espresso da una parte intera (in secondi) e da una parte decimale (in microsecondi). Anche questo parametro può essere modificato tramite il programma Preferences del Workbench.

Discussione

IND_SETTHRESH permette a un task di stabilire quale sarà il tempo di attesa prima che un tasto premuto inizi a ripetere l'evento di input generato al momento della pressione, creando in questo modo una sequenza di eventi che si ripeteranno a intervalli la cui lunghezza dipende dal valore impostato dal comando IND_SETPERIOD.

Di solito questi comandi vengono utilizzati da Intuition. I loro parametri di default si trovano nella sotto-struttura timeval appartenente alla struttura Preferences di Intuition, e possono essere modificati indirettamente tramite il programma Preferences del Workbench. Entrambi vengono inoltrati utilizzando la struttura timerequest.

IND_WRITEEVENT

Scopo del comando

Il comando IND_WRITEEVENT permette a un task di aggiungere nuove voci al flusso degli eventi di input. Questo flusso è una lista concatenata di eventi definiti dai task, generati dai dispositivi Keyboard, Gameport e Timer, o provocati dall'inserimento e dalla rimozione dei dischi. IND_WRITEEVENT utilizza il parametro io_Data della struttura IOStdReq per specificare l'indirizzo del buffer contenente le strutture InputEvent che rappresentano gli eventi da aggiungere. La lunghezza del buffer (un multiplo della dimensione della struttura InputEvent) dev'essere indicata in byte nel parametro io_Length.

IND_WRITEEVENT viene sempre eseguito in modo sincrono e ritorna alla reply port del task soltanto se il flag IOF_QUICK non è impostato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito con successo. IOERR_NOCMD indica che il task ha specificato nel parametro io_Command un comando non previsto dal dispositivo. IOERR_ABORTED indica che il comando è stato eliminato da una chiamata alla funzione AbortIO oppure dall'invio del comando CMD_FLUSH. IOERR_BADLENGTH indica che il task ha inserito nel parametro io_Length un valore non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione

del dispositivo e l'indirizzo della struttura di gestione dell'unità che il task ottiene chiamando la funzione OpenDevice.

Il task deve impostare io_Command con il comando IND_WRITEEVENT; inizializzare io_Flags a IOF_QUICK se desidera utilizzare il QuickIO, e inizializzare i seguenti parametri:

- io_Length. Deve contenere la grandezza, espressa in byte, del buffer che contiene le strutture InputEvent relative ai nuovi eventi di input.

- io_Data. Deve puntare al buffer in RAM che contiene le strutture InputEvent, ognuna delle quali rappresenta un nuovo evento di input che il dispositivo Input deve elaborare.

Discussione

Il comando IND_WRITEEVENT permette a un task di aggiungere nuovi eventi di input a un flusso di eventi. Questa possibilità può rivelarsi molto utile per simulare via software il comportamento della tastiera, del timer oppure del mouse. Può anche essere utile nel corso del debug di un programma complesso.

Il dispositivo Console

Introduzione

Il dispositivo Console viene utilizzato per inviare informazioni a una finestra di Intuition oppure per ricevere dati dalla tastiera, dai connettori delle porte giochi e dai disk drive dell'Amiga. Esso apre automaticamente il dispositivo Input, il quale a sua volta apre automaticamente i dispositivi Keyboard, Gameport e Timer. Gli eventi di input che giungono al dispositivo Console hanno generalmente origine in uno di questi tre dispositivi. Il dispositivo Console possiede una funzione di gestione degli input (si veda la lista analizzata nel precedente capitolo) dotata della priorità 0; rispetto alla funzione di Intuition, dotata della priorità 50, si vede come nell'analisi degli eventi di input venga data la precedenza a Intuition.

Contrariamente agli altri dispositivi dell'Amiga, il dispositivo Console non si avvale della struttura Unit ma della struttura ConUnit. Questa struttura viene associata a ogni unità del dispositivo al momento dell'apertura. Attraverso la sotto-struttura cu_MP di tipo MsgPort, ConUnit permette a un task di comunicare con l'unità del dispositivo, mentre attraverso il puntatore cu_Window, che individua una struttura di tipo Window, permette a un task di comunicare con la finestra di Intuition associata all'unità (cioè con le routine interne di Intuition).

Funzionamento del dispositivo Console

La Figura 8.1 (nella pagina successiva) descrive le operazioni che caratterizzano il dispositivo Console. Un task può leggere caratteri dalla tastiera dell'Amiga e nello stesso momento inviare caratteri ASCII e di controllo a una finestra di Intuition: la figura mostra il modo in cui il task riceve informazioni dai disk drive, dalla tastiera dell'Amiga e dal mouse, e dove queste informazioni vengono inviate.

Quando si apre un'unità del dispositivo Console tramite la funzione OpenDevice, all'unità viene automaticamente associata la finestra di Intuition che il task ha aperto in precedenza. OpenDevice inizializza per la gestione dell'unità una struttura ConUnit, che ha la funzione di riunire in un unico blocco di dati la struttura MsgPort dell'unità e la struttura Window della finestra di Intuition. La prima struttura viene impiegata dal dispositivo per gestire la coda alla request port dell'unità, mentre la seconda viene impiegata da Intuition per gestire la finestra. La struttura ConUnit è quindi l'intermediario per le comunicazioni tra le routine interne del dispositivo Console e le routine interne di Intuition. Il dispositivo Console accede alla struttura Window indicata dal task per conoscere le dimensioni della finestra e quindi calcolare quanti caratteri può visualizzare per riga e quante righe può visualizzare prima di far partire lo scroll.

Se un task ha necessità di utilizzare le routine interne del dispositivo Console per elaborare eventi di input provenienti dal mouse, dalla tastiera o dalle porte giochi, deve creare un'apposita reply port per ricevere le risposte ai comandi CMD_READ che invia.

Oltre a questo primo canale di comunicazione, il task deve creare un'altra reply port separata per accodare le risposte ai comandi CMD_WRITE. Se il task dovesse inviare un qualsiasi altro comando al dispositivo Console, dovrebbe inizializzare anche una terza reply port. Per creare queste message port il task può utilizzare la funzione CreatePort di supporto alla libreria Exec.

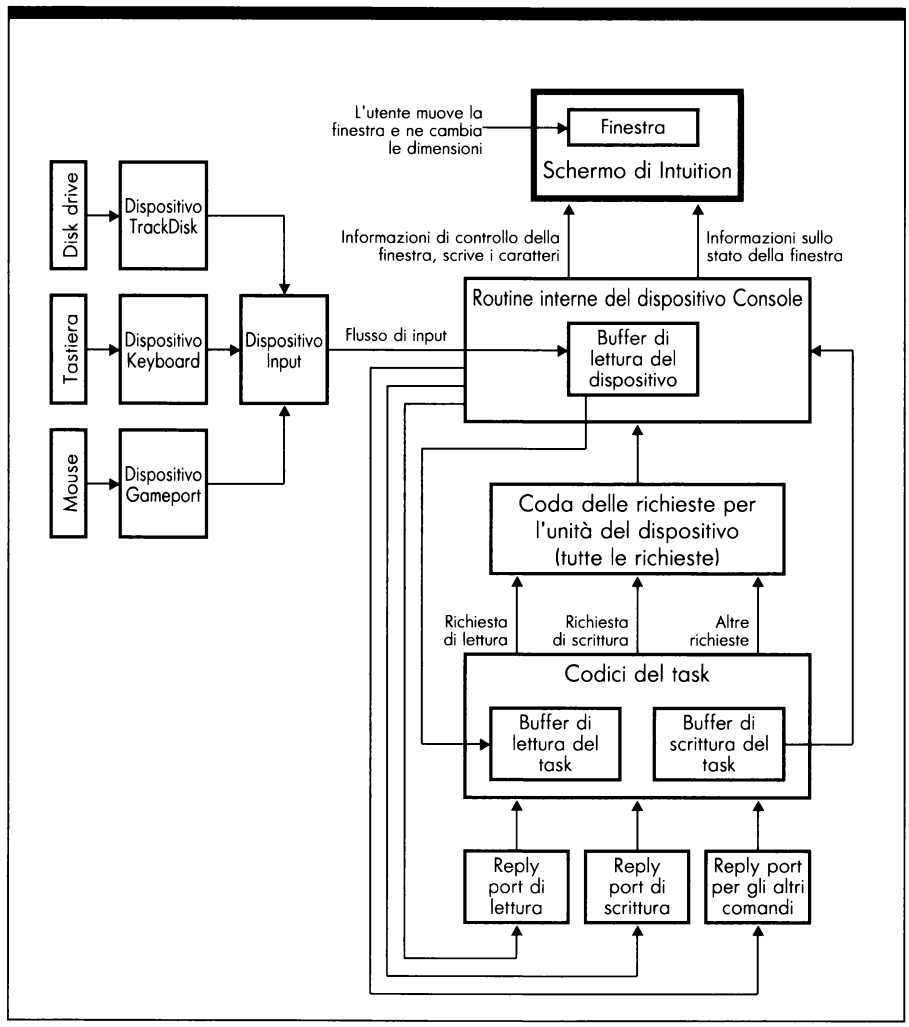


Figura 8.1: *Funzionamento del dispositivo Console*

Le operazioni di lettura e scrittura del dispositivo Console

La Figura 8.2 mostra lo schema generale delle operazioni che il dispositivo Console compie in fase di scrittura. Ogni comando CMD_WRITE inoltrato alle routine interne del dispositivo invia all'unità e alla relativa finestra di Intuition un insieme di caratteri ASCII e/o una serie di codici di controllo.

Il task deve associare una finestra di Intuition a ogni unità del dispositivo Console che apre. Questa finestra agisce da terminale ASCII avanzato: riconosce molti dei codici di controllo per la gestione dello schermo previsti dallo standard ANSI X3.64 (sequenze escape), e alcuni codici addizionali che sono propri del sistema Amiga. Oltre alle sequenze escape, l'unità aperta del dispositivo Console può inviare alla relativa finestra di Intuition caratteri ASCII, che costituiscono il testo che l'utente vede apparire sulla finestra. È compito del task definire all'interno di un proprio buffer le informazioni che vuole

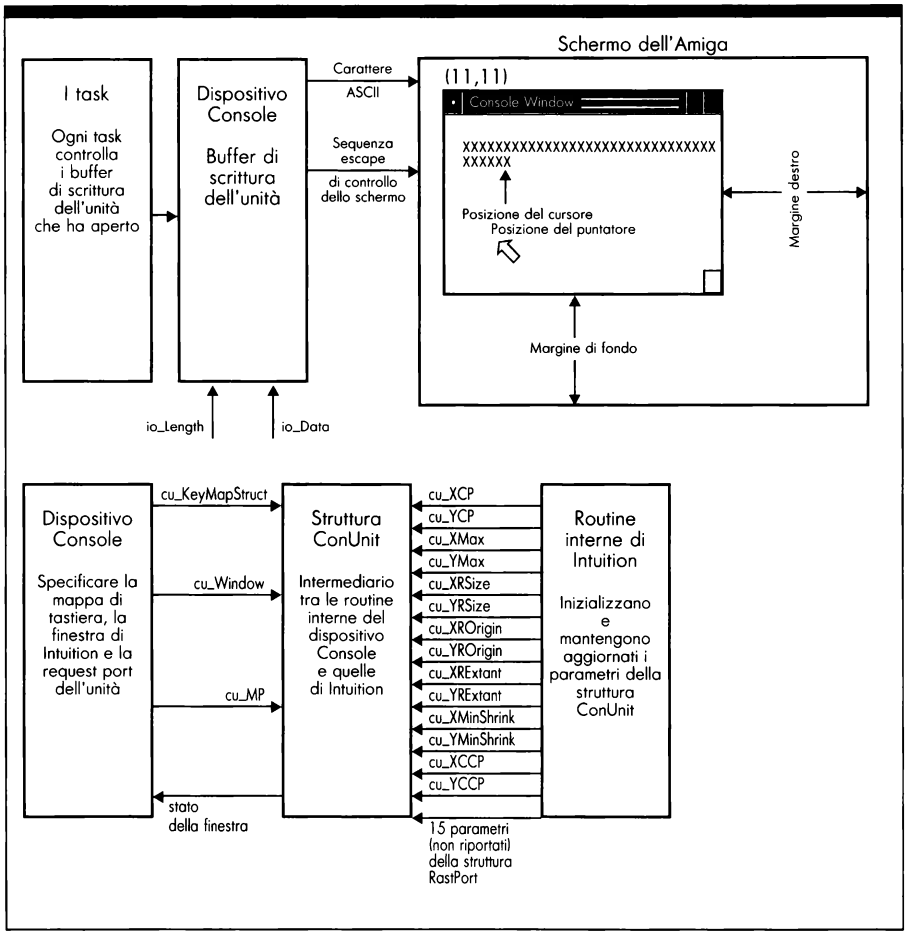


Figura 8.2: *Operazioni di scrittura del dispositivo Console*

visualizzare prima d'inviare al dispositivo un comando CMD_WRITE.

Le relazioni intercorrenti tra le routine interne del dispositivo Console e le routine interne di Intuition vengono mostrate nella parte inferiore della Figura 8.2. L'insieme di frecce che dalle routine interne di Intuition giungono alla struttura ConUnit rappresentano il percorso delle informazioni durante il trasferimento.

Le routine interne di Intuition tengono conto di ogni cambiamento riguardante la posizione e le dimensioni delle finestre, aggiornando automaticamente 14 parametri delle relative strutture ConUnit.

Questa struttura permette a un task e alle routine interne del dispositivo Console di tenere sempre sotto controllo lo stato di una finestra. Un task può accedere in lettura a tutti i 14 parametri della struttura ConUnit, ma non può alterarli. Le routine interne del dispositivo Console utilizzano infatti questi parametri per determinare il modo in cui devono visualizzare il testo nella finestra. Un evento che invece produce alterazioni in questi parametri è la manipolazione della finestra effettuata dall'utente con il mouse. Oltre ai parametri citati, nella struttura ConUnit sono presenti altri 15 parametri che vengono aggiornati con valori contenuti nella struttura RastPort, una delle più importanti strutture gestite da Intuition.

Le routine del dispositivo Console ricevono comandi dal task attraverso una message port definita dalla sotto-struttura cu_MP, di tipo MsgPort, appartenente alla struttura ConUnit. Essa costituisce la request port dell'unità e ne definisce la coda. Il parametro cu_Window della struttura ConUnit viene aggiornato dalla funzione OpenDevice con l'indirizzo della struttura Window che il task indica nella struttura di I/O (si ricordi che la finestra dev'essere già stata aperta dal task). Il parametro cu_KeyMapStruct rappresenta il nome della struttura KeyMap utilizzata per interpretare i dati provenienti dalla tastiera durante l'elaborazione dei comandi CMD_READ. Questa struttura viene inizializzata con una mappa di tastiera di default, che il task può comunque alterare.

La Figura 8.3 (nella pagina successiva) illustra le operazioni compiute dal dispositivo Console per eseguire un'operazione di lettura. Ogni comando CMD_READ inviato al dispositivo Console legge dal buffer interno del dispositivo uno dei seguenti tipi d'informazioni e le trasferisce nel buffer di lettura del task.

- Un flusso di caratteri ANSI X3.64 provenienti dalla tastiera attraverso i dispositivi Keyboard e Input. Questo flusso può contenere caratteri ASCII oppure informazioni su eventi di input allo stato grezzo.

- Un flusso di dati provenienti dal mouse dell'Amiga attraverso i dispositivi Gameport e Input quando l'utente agisce con il mouse all'interno di una finestra di Intuition.

- Un evento di input relativo all'inserimento o alla rimozione di un disco dal disk drive, proveniente dai dispositivi TrackDisk e Input.

Il dispositivo Console può lavorare con due tipi di input: gli eventi grezzi

e gli eventi preelaborati. Un programma musicale, per esempio, può interagire con eventi da tastiera allo stato grezzo, che non abbiano cioè subìto alcuna elaborazione sulla base della mappa di tastiera attiva. Al contrario, un word processor può desiderare che gli eventi di input provenienti dalla tastiera siano preelaborati (trasformati in codici ASCII e sequenze escape).

Tutti e tre i tipi di eventi di input che abbiamo illustrato possono essere grezzi (se al dispositivo giunge la sequenza escape SRE: set raw events, non preelaborare gli eventi grezzi) oppure preelaborati (se al dispositivo giunge la sequenza escape RRE: reset raw events, preelabora gli eventi grezzi).

Ogni evento di input è originariamente rappresentato da una struttura InputEvent; prima di raggiungere il dispositivo Console gli eventi vengono riuniti in un unico flusso dalle routine del dispositivo Input (si veda il capitolo 7 per maggiori dettagli sulle operazioni compiute dal dispositivo Input).

Se è attiva la sequenza RRE, il flusso di input di tastiera viene preelaborato tramite una mappa di tastiera prima che i singoli caratteri giungano al buffer interno del dispositivo Console. La mappa di tastiera associa a ogni tasto un carattere o una stringa di caratteri; il flusso di questi nuovi dati viene infine trasferito nel buffer che il task indica nella struttura di I/O del comando CMD_READ. Il sistema prevede per default la mappa di tastiera americana, ma l'utente può abilitarne una qualsiasi. Il dispositivo Keyboard mette a disposizione la struttura KeyMap e i comandi CD_ASKKEYMAP e CD_SETKEYMAP per gestire la mappa.

L'Amiga prevede attualmente la possibilità di avere una tastiera tedesca, spagnola, francese, inglese, italiana, islandese, svedese, danese, norvegese, canadese e americana. A queste aggiunge una mappa di tastiera americana compatibile con la release 1.1 del software sistema, e una mappa di tastiera che cambia la tastiera dallo standard QWERTY a quello DVORAK. Queste mappe si trovano nella directory devs/keymaps del disco Workbench. L'utente seleziona la mappa desiderata attivando il programma Setmap tramite la sua icona o impartendo direttamente da CLI il comando SETMAP.

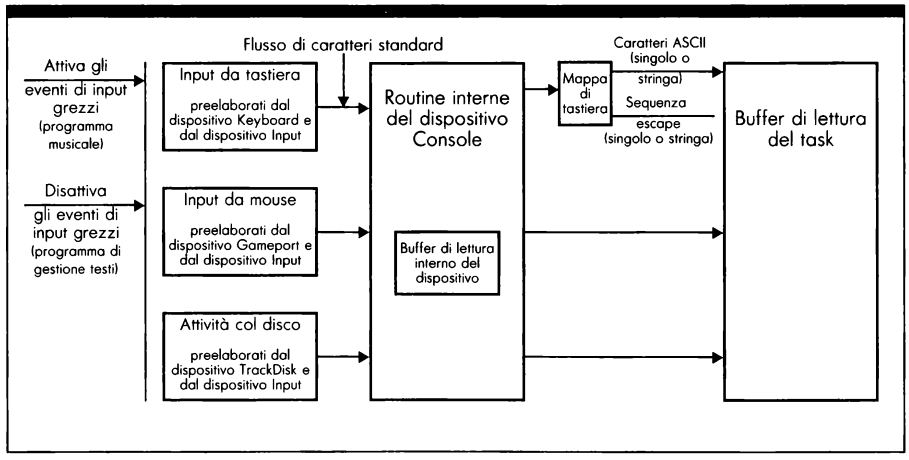


Figura 8.3: *Operazioni di lettura del dispositivo Console*

Una volta elaborato, il flusso di caratteri viene suddiviso in due flussi: uno di soli caratteri ASCII e l'altro costituito da un insieme di caratteri utilizzati per definire sequenze escape composte da un singolo carattere o da una stringa di caratteri preceduta dal carattere escape. Entrambi questi flussi vengono memorizzati nel buffer di lettura del dispositivo Console per la successiva elaborazione da parte del task. Per esempio, il task potrebbe ritrasmettere i dati a un'altra finestra sullo schermo inviando a quell'unità un comando CMD_WRITE e indicando come buffer quello che ha impiegato per ricevere i dati dalla prima finestra.

Gli eventi di input del mouse vengono inviati direttamente dal dispositivo Gameport alle routine del dispositivo Console, che li elaborano. Questi eventi di input danno origine quasi sempre a una serie di azioni nella finestra di Intuition.

Gli eventi provocati dall'inserimento e dalla rimozione del disco dal disk drive vengono inviati direttamente da TrackDisk alle routine del dispositivo Console, che li elaborano. Anche questi eventi possono dare origine a qualche azione nella finestra di Intuition (per esempio, l'apparizione di un requester che chieda all'utente d'inserire un certo disco).

comandi del dispositivo Console

Il dispositivo Console prevede quattro comandi specifici, tre comandi standard e due funzioni dedicate, non comuni agli altri dispositivi. Tutti i comandi possono essere inviati richiedendo il QuickIO. Nessun comando viene eseguito in modo immediato. Tutti i comandi influenzano il parametro io_Error della struttura IOStdReq; CMD_READ, inoltre, influenza il parametro io_Actual e il contenuto del buffer interno di lettura del dispositivo.

L'interazione con il dispositivo Console prevede tre fasi.

1. *Preparazione della struttura IOStdReq*. Il programmatore ha un completo controllo in questa fase: qui vengono inizializzati i parametri della struttura IOStdReq, necessari per inviare un comando alle routine interne del dispositivo o per chiamare una funzione. Questi parametri includono quelli utilizzati dalla maggior parte dei dispositivi, nonché i particolari parametri richiesti dalle funzioni CDInputHandler e RawKeyConvert; la scelta dei parametri dipende dal tipo di comando che s'intende inviare o dalla funzione che si desidera chiamare.

2. *Invio della richiesta e sua elaborazione*. L'unico compito del task in questa fase è quello d'inviare il comando al dispositivo utilizzando le funzioni BeginIO, DoIO o SendIO. Poi il controllo passa al sistema e alle routine interne del dispositivo.

3. *Elaborazione dei parametri di output del comando e loro restituzione*. In questa fase il sistema e le routine interne del dispositivo possiedono un controllo completo. I risultati prodotti dall'elaborazione della richiesta

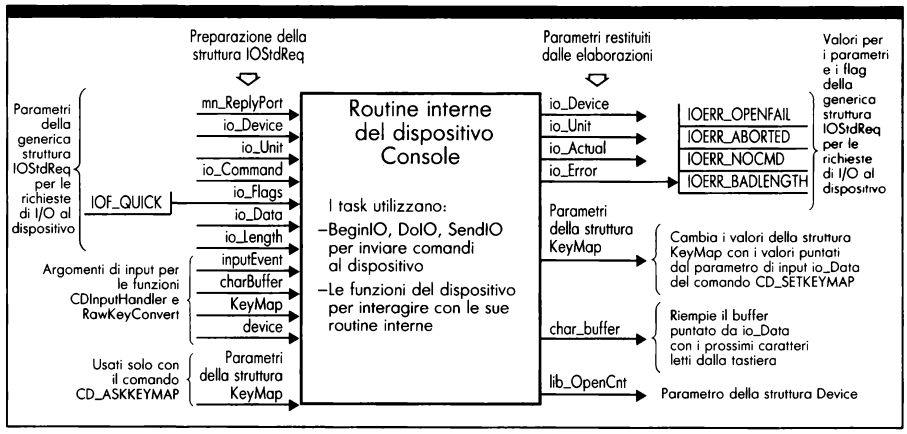


Figura 8.4: *Gestione delle funzioni e dei comandi previsti dal dispositivo Console*

vengono restituiti al task che l'ha inviata. Se la richiesta non prevede il QuickIO, il comando deve attendere di giungere alla sommità della coda alla request port, e successivamente viene elaborato e inserito nella coda alla reply port del task. Nel caso che invece venga richiesto il QuickIO, non si verifica nessun accodamento alla request port e la richiesta giunge direttamente alle routine interne del dispositivo.

Per la maggior parte dei comandi del dispositivo Console, il sistema restituisce un valore nel parametro io_Error; per il comando CMD_READ restituisce un valore anche nel parametro io_Actual. Inoltre, i comandi CD_ASKKEYMAP e CD_SETKEYMAP rispettivamente restituiscono e modificano i dati che nella struttura KeyMap costituiscono la mappa di tastiera.

La Figura 8.4 mostra anche i parametri che intervengono nell'inizializzazione e nella gestione del dispositivo Console. Le funzioni OpenDevice e CloseDevice influenzano il parametro lib_OpenCnt della struttura Device: OpenDevice, inoltre, influenza anche il parametro io_Error. Si tenga presente. infine, che ConUnit non contiene un parametro come unit_OpenCnt, che tenga il conto delle volte che la struttura è stata aperta e non ancora chiusa. Il motivo è semplice: nel dispositivo Console le unità sono ad accesso esclusivo, cioè possono essere aperte solo da un task alla volta. Inoltre, a differenza di altri dispositivi, Console è in grado di aprire e di gestire un numero virtualmente infinito di unità, e quindi due task non si trovano mai nella necessità di aprire contemporaneamente la stessa unità.

Le strutture del dispositivo Console

La Figura 8.5 (nella pagina successiva) mostra le strutture previste dal dispositivo Console. Questo dispositivo comunica direttamente solo con la struttura ConUnit, tuttavia prevede anche altre tre strutture per interagire con

la tastiera dell'Amiga: KeyMapNode, KeyMap e KeyMapResource.

ConUnit contiene due sotto-strutture, la struttura cu_MP di tipo MsgPort, e la struttura cu_KeyMapStruct di tipo KeyMap. La struttura MsgPort rappresenta la request port dell'unità e individua in memoria la relativa coda (la struttura KeyMap viene discussa nel capitolo 9).

ConUnit contiene inoltre tre puntatori. Il puntatore cu_Window deve individuare in memoria la struttura Window che definisce la finestra associata all'unità (che dev'essere già stata aperta quando il task chiama OpenDevice; è infatti questa funzione che si preoccupa di aggiornare il puntatore cu_Window). Il puntatore cu_AreaPtrn individua l'area RAM contenente la matrice grafica di riempimento definita dalla libreria Graphics (si veda il capitolo 2 del volume I). Infine, il puntatore cu_Font individua la struttura TextFont (si veda il capitolo 4 del volume I) che indica quale fonte-carattere dev'essere utilizzata. Questi puntatori contribuiscono alla gestione della finestra di Intuition associata all'unità.

La struttura KeyMap non contiene sotto-strutture, ma solo una serie di otto puntatori. Ognuno individua una diversa area RAM nella quale vengono mentenute le informazioni relative a una specifica mappa di tastiera.

La struttura KeyMapNode contiene due sotto-strutture, una denominata kn_Node (di tipo Node) e una denominata kn_KeyMap (di tipo KeyMap). Il sistema utilizza la struttura Node per inserire una struttura KeyMapNode, e quindi anche la relativa sotto-struttura KeyMap, in una lista mantenuta dal sistema.

La struttura KeyMapResource contiene due sotto-strutture, una denominata kr_Node (di tipo Node) e una denominata kr_List (di tipo List). Il sistema le

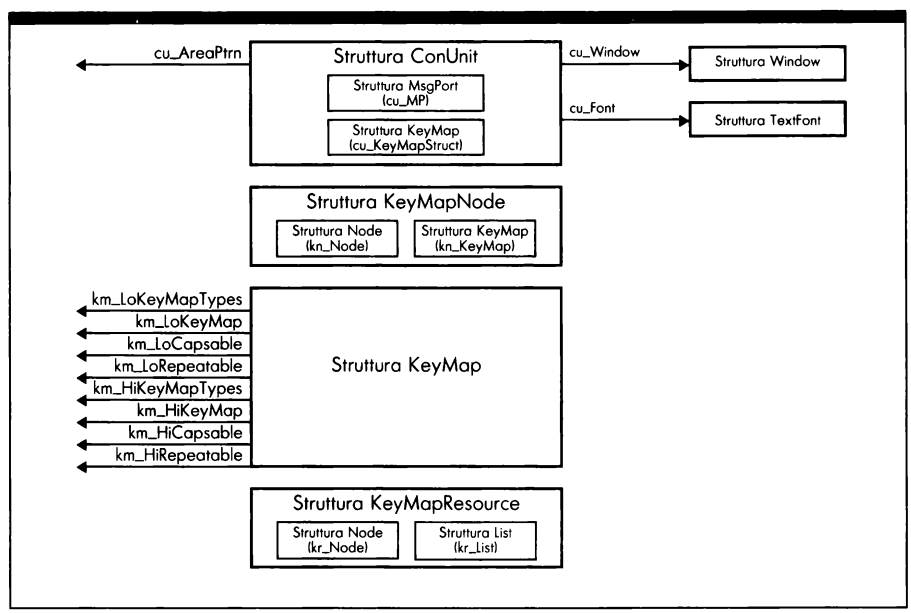


Figura 8.5: *Strutture utilizzate dal dispositivo Console*

utilizza per mantenere una lista delle risorse di tastiera disponibili nell'intero sistema.

La struttura ConUnit

La struttura ConUnit è definita come segue:

```
struct ConUnit {
        struct MsgPort cu_MP;
        struct Window *cu_Window;
        WORD cu_XCP;
        WORD cu_YCP;
        WORD cu_XMax;
        WORD cu_YMax;
        WORD cu_XRSize;
        WORD cu_YRSize;
        WORD cu_XROrigin;
        WORD cu_YROrigin;
        WORD cu_XRExtant;
        WORD cu_YRExtant;
        WORD cu_XMinShrink;
        WORD cu_YMinShrink;
        WORD cu_XCCP;
        WORD cu_YCCP;
        struct KeyMap cu_KeyMapStruct;
        UWORD cu_TabStops[MAXTABS];
        BYTE cu_Mask;
        BYTE cu_FgPen;
        BYTE cu_BgPen;
        BYTE cu_AOLPen;
        BYTE cu_DrawMode;
        BYTE cu_AreaPtSz;
        APTR cu_AreaPtrn;
        UBYTE cu_Minterms[8];
        struct TextFont *cu_Font;
        UBYTE cu_AlgoStyle;
        UBYTE cu_TxFlags;
        UWORD cu_TxHeight;
        UWORD cu_TxWidth;
        UWORD cu_TxBaseline;
        UWORD cu_TxSpacing;
        UBYTE cu_Modes[(PMB_AWM+7)/8];
        UBYTE cu_RawEvents[(IECLASS_MAX+7)/8];
};
```

- cu_MP è il nome della sotto-struttura MsgPort che rappresenta la coda alla request port dell'unità.

- cu_Window punta alla sotto-struttura Window che rappresenta la finestra di Intuition associata all'unità.

Ai successivi 14 parametri il task può accedere solo in lettura. Questi parametri vengono inizializzati durante l'esecuzione di OpenDevice e vengono mantenuti automaticamente aggiornati dalle routine interne di Intuition in modo da rispecchiare sempre le condizioni della finestra. Si noti che tutte le dimensioni sono relative all'angolo superiore sinistro della finestra, e che le coordinate X e Y delle posizioni dei caratteri e del cursore sono espresse rispettivamente in larghezza e altezza del carattere (per esempio, X = 5 e Y = 0 indica la posizione che si otterrebbe spostando il cursore a destra di cinque caratteri sulla prima riga della finestra).

- cu_XCP e cu_YCP rappresentano le coordinate X e Y dell'ultimo carattere visualizzato nella finestra di Intuition.

- cu_XMax e cu_YMax rappresentano le massime coordinate X e Y alle quali un carattere può essere visualizzato.

- cu_XRSize e cu_YRSize indicano il massimo numero di caratteri che possono essere visualizzati nella finestra, in direzione rispettivamente orizzontale e verticale. Questi parametri vengono utilizzati nelle operazioni di wordwrap (la procedura per mandare a capo una parola quando una delle sue lettere uscirebbe dal margine destro della finestra) e per l'impaginazione.

- cu_XROrigin e cu_YROrigin indicano l'origine delle coordinate X e Y nella finestra associata alla struttura ConUnit.

- cu_XRExtant e cu_YRExtant indicano le dimensioni massime (in orizzontale e in verticale) dell'area di memoria allocata per contenere la finestra. Quest'area di memoria viene chiamata raster e contiene la bitmap della finestra associata all'unità del dispositivo Console. I valori presenti in questi due parametri sono espressi in pixel.

- cu_XMinShrink e cu_YMinShrink indicano le dimensioni minime che la finestra di Intuition può raggiungere quando l'utente (o il task) la rimpicciolisce.

- cu_XCCP e cu_YCCP indicano la posizione del cursore nella finestra. Vengono aggiornate a mano a mano che l'utente lo muove.

Ai due successivi parametri della struttura ConUnit, il task può invece accedere sia in lettura sia in scrittura.

- cu_KeyMapStruct è il nome della sotto-struttura KeyMap utilizzata dall'unità del dispositivo Console per interpretare i codici provenienti dalla tastiera. Il task può interagire con la struttura KeyMap tramite i comandi CD_ASKKEYMAP e CD_SETKEYMAP.

- cu_TabStops[MAXTABS] è un insieme di word che rappresentano i tabulatori della finestra di Intuition; il massimo numero di tabulatori ammesso è 80.

I 15 parametri che seguono vengono inizializzati da OpenDevice con valori contenuti nella struttura RastPort della libreria Graphics (utilizzata per le operazioni di disegno e scrittura nelle finestre di Intuition). Variando questi parametri, per esempio prima di un comando CMD_WRITE, il task muta il risultato che si ottiene sullo schermo. Si veda il capitolo 2 del volume I per una descrizione dell'impiego di questi parametri. Eccone un breve sommario:

- cu_Mask indica quali piani di bit di una bitmap verranno interessati da un'operazione di scrittura.

- cu_FgPen indica il colore del tratto sullo schermo principale (foreground screen).

- cu_BgPen indica il colore del tratto sullo schermo di fondo (background screen).

- cu_AOLPen indica il colore del tratto per i contorni delle aree.

- cu_DrawMode indica in che modo viene prodotto il tratto della penna.

- cu_AreaPtSz indica la grandezza (espressa in word) della matrice grafica di riempimento delle aree.

- cu_AreaPtrn indica l'indirizzo in memoria della matrice grafica di riempimento delle aree.

- cu_Minterms[8] è un parametro composto da 8 byte impiegati per la gestione del Blitter.

- cu_Font è un puntatore a una struttura TextFont associata con la struttura RastPort.

- cu_AlgoStyle indica quale algoritmo dev'essere impiegato per rappresentare i caratteri con un particolare stile (nero, corsivo...).

- cu_TxFlags indica i flag relativi al testo.

- cu_TxHeight indica l'altezza del carattere.

- cu_TxWidth indica la larghezza del carattere.
- cu_TxBaseline indica la linea di base del carattere.
- cu_TxSpacing indica la spaziatura prevista dalla fonte-carattere selezionata.

Gli ultimi due parametri della struttura ConUnit sono riservati al sistema.

- cu_Modes[(PMB_AWM+7)/8] individua otto modi possibili di funzionamento per l'unità. Ogni bit rappresenta un modo. Il parametro viene utilizzato internamente dal dispositivo.
- cu_RawEvents[(IECLASS_MAX+7)/8] è una serie di "deviatori" per gli eventi di input grezzi. Questo numero è legato al numero massimo di classi nelle quali è possibile suddividere gli eventi di input allo stato grezzo; viene utilizzato internamente dal dispositivo.

La struttura KeyMap

La struttura KeyMap è definita come segue:

```
struct KeyMap {
        UBYTE *km_LoKeyMapTypes;
        ULONG *km_LoKeyMap;
        UBYTE *km_LoCapsable;
        UBYTE *km_LoRepeatable;
        UBYTE *km_HiKeyMapTypes;
        ULONG *km_HiKeyMap;
        UBYTE *km_HiCapsable;
        UBYTE *km_HiRepeatable;
};
```

Prima di procedere alla spiegazione di ogni parametro, affrontiamo senza entrare nei dettagli la gestione della tastiera.

A ogni tasto è associato un codice grezzo. Si tratta di un valore numerico, che prescinde dal significato che si può attribuire al tasto via software. Tramite questi codici grezzi è possibile individuare univocamente la pressione di ogni singolo tasto. Per esempio, quando l'utente preme il tasto Shift di sinistra, la tastiera restituisce il valore 0x60. La tastiera segnala anche il rilascio di un tasto, restituendone il codice grezzo con il bit più significativo impostato.

L'insieme dei tasti, e quindi dei relativi codici grezzi, viene suddiviso in due sottoinsiemi: i codici della tastiera bassa e i codici della tastiera alta. Nel primo insieme sono compresi i codici che vanno dal valore 0x00 al valore 0x3F (si tratta di tutti i tasti alfabetici, dei tasti numerici e di alcuni segni d'interpunzione). Nel secondo insieme sono compresi i codici che vanno dal valore 0x40 al valore 0x67; si tratta dei tasti funzione, della barra spaziatrice,

del Return, dei due tasti Shift, delle frecce di spostamento del cursore, dei tasti Ctrl, Alt, Tab, Esc e altri ancora.

Nella struttura KeyMap sono contenuti otto puntatori, di cui i primi quattro individuano dati relativi alla tastiera bassa e gli altri quattro gli stessi dati per la tastiera alta. Vediamo i loro significati.

- km_LoKeyMapTypes punta a una tavola di byte, uno per ogni tasto della tastiera bassa. Questi byte indicano come dev'essere interpretata la serie di quattro byte associata a ogni singolo tasto.

- km_LoKeyMap punta alla tavola di decodifica relativa ai tasti della tastiera bassa. Ogni elemento della tavola è lungo quattro byte. Ci sono due possibili modi per interpretare il significato di questa serie di byte (la scelta del modo dipende dal byte nella tavola KeyMapTypes appena illustrata). Il primo modo è quello di considerare ogni byte singolarmente: il primo rappresenta il carattere associato al tasto quando è premuto da solo, il secondo quando il tasto è premuto insieme al primo qualificatore, il terzo quando il tasto è premuto insieme al secondo qualificatore, il quarto quando il tasto è premuto insieme a entrambi i qualificatori (si possono considerare "tasti qualificatori" i tasti Ctrl, Alt e Shift). L'altro modo d'interpretazione è quello d'individuare nei quattro byte una long word contenente l'indirizzo di una stringa di caratteri associata al tasto. Sfruttando a fondo questa organizzazione, i programmatori possono creare gestioni della tastiera davvero sofisticate.

- km_LoCapsable punta a una tavola di 8 byte (64 bit) contenente informazioni sul processo d'interpretazione dei codici grezzi generati dalla pressione dei tasti sulla tastiera; essa indica al sistema per quali tasti della tastiera bassa abilitare il tasto Caps Lock equivale a premere il tasto Shift. In pratica, la tavola individua i tasti per i quali l'interpretazione cambia qualora il tasto Caps Lock sia abilitato. Ogni bit rappresenta un particolare tasto. I bit sono numerati con il seguente criterio: il bit 0 del byte 0 corrisponde al codice grezzo 0x00, il bit 7 del byte 0 corrisponde al codice grezzo 0x07, il bit 0 del byte 1 corrisponde al codice grezzo 0x08, e così via.

- km_LoRepeatable punta a una tavola di 8 byte (64 bit) che indica al sistema quali tasti della tastiera bassa devono essere ripetuti quando l'utente li mantiene premuti oltre il periodo di tempo impostato dal programma Preferences, o dal comando IND_SETTHRESH del dispositivo Input. I bit di questa tavola seguono la stessa numerazione illustrata per la tavola precedente.

I quattro parametri seguenti svolgono le stesse funzioni dei precedenti, ma in riferimento alla tastiera alta.

- km_HiKeyMapTypes punta a una tavola di byte, uno per ogni tasto della

tastiera alta. Questi byte indicano come dev'essere interpretata la serie di quattro byte associata a ogni singolo tasto.

- km_HiKeyMap punta alla tavola di decodifica relativa ai tasti della tastiera alta.

- km_HiCapsable punta a una tavola di 8 byte (64 bit) contenente informazioni sul processo d'interpretazione dei codici grezzi generati dalla pressione dei tasti sulla tastiera; essa indica al sistema per quali tasti della tastiera alta abilitare il tasto Caps Lock equivale a premere il tasto Shift. In pratica, la tavola individua i tasti per i quali l'interpretazione cambia qualora il tasto Caps Lock sia abilitato. Ogni bit rappresenta un particolare tasto. I bit sono numerati con il seguente criterio: il bit 0 del byte 0 corrisponde al codice grezzo 0x40, il bit 7 del byte 0 corrisponde al codice grezzo 0x47, il bit 0 del byte 1 corrisponde al codice grezzo 0x48, e così via.

- km_HiRepeatable punta a una tavola di 8 byte (64 bit) che indica al sistema quali tasti della tastiera alta devono essere ripetuti quando l'utente li mantiene premuti oltre il periodo di tempo impostato dal programma Preferences, o dal comando IND_SETTHRESH del dispositivo Input. I bit di questa tavola seguono la stessa numerazione illustrata per la tavola precedente.

La struttura KeyMapNode

La struttura KeyMapNode è definita come segue:

```
struct KeyMapNode {
        struct Node kn_Node;
        struct KeyMap kn_KeyMap;
};
```

I parametri della struttura KeyMapNode hanno i seguenti significati:

- kn_Node è il nome della sotto-struttura Node utilizzata per concatenare un insieme di strutture KeyMap in una lista.

- kn_KeyMap è il nome della sotto-struttura KeyMap che dev'essere inserita nella lista delle strutture KeyMap mantenuta dal sistema.

La struttura KeyMapResource

La struttura KeyMapResource è definita come segue:

```
struct KeyMapResource {
```

```
    struct Node kr_Node;
    struct List kr_List;
};
```

I parametri della struttura KeyMapResource hanno i seguenti significati:

- kr_Node è il nome della sotto-struttura Node utilizzata per concatenare in un'unica lista le strutture KeyMapResource.
- kr_List è il nome della sotto-struttura List utilizzata per mantenere la lista delle strutture KeyMap.

IMPIEGO DELLE FUNZIONI

CDInputHandler

Sintassi di chiamata della funzione

```
newInputEvent = CDInputHandler (oldInputEvent, device)
D0                              A0          A1
```

Scopo della funzione

Questa funzione gestisce gli eventi di input che interessano il dispositivo Console. Il task di input presente in ROM genera un flusso di eventi, che in pratica è una lista semplice di strutture InputEvent. L'indirizzo della prima struttura InputEvent viene passato come argomento alla funzione CDInputHandler, in modo che elabori gli eventi relativi al dispositivo Console. Gli eventi di input non elaborati da CDInputHandler vengono passati alla lista di funzioni di gestione degli input illustrata nel precedente capitolo.

CDInputHandler restituisce nella variabile newInputEvent l'indirizzo di una struttura InputEvent. Questa struttura, oltre che descrivere il primo evento del flusso creato dal dispositivo Input, costituisce anche il primo elemento della lista che si forma dopo l'elaborazione condotta dalla funzione CDInputHandler. Questa lista, o flusso, viene quindi inviata per ulteriori elaborazioni alle funzioni di gestione degli input. Nella lista, ogni evento di input è concatenato al successivo tramite il parametro ie_NextEvent della struttura InputEvent che lo caratterizza.

CDInputHandler è inclusa anche nella release 1.2 del software sistema per assicurare la compatibilità con i programmi creati per le versioni precedenti del sistema operativo. Un programma creato per la release 1.2 e successive non dovrebbe utilizzare la funzione CDInputHandler, e al suo posto dovrebbe avvalersi delle funzioni di gestione degli input definibili tramite il dispositivo Input, com'è stato descritto nel capitolo 7.

Argomenti della funzione

oldInputEvent Deve contenere l'indirizzo di una struttura InputEvent che rappresenta il primo evento di input in una lista di eventi concatenati.

device Deve contenere l'indirizzo della struttura Device ottenuto durante l'apertura del dispositivo tramite la funzione OpenDevice. In pratica individua l'indirizzo base della libreria di routine che costituisce il dispositivo Console.

Preparazione della struttura IOStdReq

La funzione CDInputHandler non richiede come argomento l'indirizzo di una struttura di I/O.

Discussione

CDInputHandler è l'unica funzione del dispositivo Console che elabora direttamente gli eventi di input. Essa elabora la lista delle strutture InputEvent (cioè il flusso di eventi generato dal dispositivo Input) tra loro concatenate tramite il parametro ie_NextEvent. Gli eventi di input che non sono stati rimossi dalla funzione CDInputHandler vengono quindi inviati alle funzioni di gestione degli input previste dal dispositivo Input. L'indirizzo newInputEvent restituito dalla funzione CDInputHandler individua la prima struttura InputEvent nella nuova lista di eventi che si forma dopo l'elaborazione. Questa lista corrisponde alla lista originale scremata degli eventi di input che CDInputHandler ha rimosso.

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOStdReq)
A1

Scopo della funzione

Questa funzione chiude l'accesso da parte del task a un'unità del dispositivo Console. Se si tratta della chiamata alla funzione CloseDevice che chiude l'ultima unità aperta dal task (e il dispositivo Input risulta chiuso per il task) automaticamente vengono chiusi, sempre per il task, anche i dispositivi Timer, Keyboard e Gameport. Quando CloseDevice restituisce il controllo, il task non può più interagire con l'unità chiusa. Nel caso che, con la chiamata alla funzione CloseDevice, tutte le unità del dispositivo che il task aveva aperto risultino chiuse, la funzione imposta i parametri io_Device e io_Unit della struttura IOStdReq a −1; il task non più utilizzare questa struttura IOStdReq fino a quando i parametri in essa contenuti non vengono nuovamente inizializzati da OpenDevice. Inoltre, in questo caso, CloseDevice decrementa di 1 anche il parametro lib_OpenCnt della struttura Device.

Argomenti della funzione

iOStdReq — Deve contenere l'indirizzo della struttura di tipo IOStdReq inizializzata da OpenDevice.

Discussione

CloseDevice chiude per il task l'accesso all'unità del dispositivo Console indicata dal parametro io_Unit della struttura di I/O. La finestra di Intuition aperta dal task e associata all'unità dalla funzione OpenDevice dev'essere chiusa dal task tramite CloseWindow; il dispositivo Console, infatti, si preoccupa soltanto di chiudere l'unità, liberando la memoria occupata dalla struttura ConUnit. Eventualmente, il task può impiegare la finestra per altri scopi.

Prima di chiamare CloseDevice, un task dovrebbe sempre verificare se tutte le richieste di I/O inviate alle finestre di I/O sono state restituite dalle

routine del dispositivo Console, cioè, se esistono delle risposte accodate alla sua reply port. Per effettuare questo controllo si possono utilizzare le funzioni GetMsg, Remove, CheckIO e WaitIO.

Se il task apre il dispositivo indicando come argomento l'unità −1, OpenDevice restituisce l'indirizzo della struttura Device di gestione del dispositivo e imposta a −1 il puntatore io_Device per indicare che nessuna unità è stata aperta. In questo caso CloseDevice si limita ovviamente alla semplice chiusura del dispositivo.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("console.device", unit, iOStdReq, ØL)
DØ                 AØ                DØ  A1       D1
```

Scopo della funzione

Questa funzione apre l'accesso al dispositivo Console ed eventualmente a un'unità. OpenDevice apre anche il dispositivo Input, il quale a sua volta apre i dispositivi Timer, Gameport e Keyboard nel caso in cui non siano già stati aperti dal task.

Se viene indicato nell'argomento unit della chiamata il valore −1, OpenDevice non apre nessuna unità e restituisce l'indirizzo della struttura Device di gestione del dispositivo; questo indirizzo, memorizzato nel parametro io_Device della struttura di I/O, può essere utilizzato come argomento nelle chiamate alle funzioni CDInputHandler e RawKeyConvert (le quali richiedono espressamente l'indirizzo base del dispositivo per interagire con le sue routine interne). Se invece viene indicato nell'argomento unit il valore 0, OpenDevice apre un'unità e le associa la finestra di Intuition aperta in precedenza.

Impostare a 0 l'argomento unit della chiamata non significa chiamare l'unità 0, ma soltanto segnalare che si desidera aprire un'unità. Il dispositivo Console infatti è in grado di aprire un numero virtualmente infinito di unità. L'argomento unit agisce quindi come flag: ogni volta che vale 0, chiamando OpenDevice viene aperta una nuova unità.

Quando l'argomento unit contiene il valore 0, la funzione OpenDevice alloca e inizializza automaticamente una struttura ConUnit che viene impiegata per la gestione dell'unità aperta. ConUnit è una struttura completamente diversa dalla struttura Unit prevista da altri dispositivi. Contiene una sotto-struttura MsgPort che rappresenta la coda alla request port dell'unità e un puntatore inizializzato da OpenDevice con l'indirizzo della

struttura Window della finestra. OpenDevice, inoltre, incrementa di 1 il parametro lib_OpenCnt della struttura Device.

Le routine del dispositivo Console danno per scontato che la libreria di Intuition e la finestra siano aperte nel momento in cui viene chiamata OpenDevice con unit = 0. Come parte della preparazione della chiamata, il parametro io_Data della struttura IOStdReq dev'essere inizializzato con l'indirizzo della struttura Window che il task ottiene chiamando la funzione OpenWindow. La struttura RastPort associata con la finestra, e quindi la stessa finestra, può essere già stata utilizzata per altri impieghi prima di essere associata all'unità del dispositivo Console (per ottenere maggiori dettagli sulla struttura RastPort, si veda il capitolo 6 del Volume I).

Ogni unità del dispositivo Console viene aperta nel modo di accesso esclusivo e viene associata con una sola finestra di Intuition per volta. Tuttavia, le routine interne del dispositivo Console vengono condivise tra tutte le unità aperte dai task.

I risultati prodotti dall'esecuzione della funzione sono i seguenti:

- io_Device. OpenDevice memorizza in questo parametro l'indirizzo della struttura Device di gestione del dispositivo.

- io_Unit. OpenDevice memorizza in questo parametro l'indirizzo della struttura ConUnit, che crea e inizializza se nell'argomento unit è stato passato il valore 0. OpenDevice assegna a ogni nuova unità del dispositivo Console un'identica sotto-struttura ConUnit. Se invece il task chiama OpenDevice indicando il valore −1, la funzione memorizza nel parametro io_Unit il valore −1.

- io_Error. Il valore 0 indica che la richiesta di apertura è stata eseguita. Il codice d'errore IOERR_OPENFAIL segnala che il dispositivo Console non può essere aperto; in genere questo errore dipende dal fatto che nel sistema non c'è sufficiente memoria.

Argomenti della funzione

"console.device" Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Console.

unit Il task deve indicare in questo argomento il valore −1 se non desidera aprire nessuna unità, altrimenti deve indicare il valore 0.

iOStdReq Il task deve indicare in questo argomento l'indirizzo della struttura di tipo IOStdReq che intende impiegare per interagire con il dispositivo.

ØL Indica che la funzione ignora l'argomento flag.

Preparazione delle struttura IOStdReq

Si deve inizializzare mn_ReplyPort in modo che punti a una struttura MsgPort del task (ovvero a una reply port). Se si desidera aprire un'unità (unit = 0) occorre memorizzare nel parametro io_Data l'indirizzo della struttura Window che definisce la finestra di Intuition da associare all'unità, e nel parametro io_Length la dimensione della struttura.

Discussione

La funzione OpenDevice apre per il task l'accesso al dispositivo Console. Se l'apertura del dispositivo comporta anche l'apertura di un'unità, OpenDevice oltre a inizializzare il parametro io_Device, inizializza una struttura di tipo ConUnit e ne memorizza l'indirizzo nel parametro io_Unit. Questo indirizzo dovrà essere sempre indicato nelle richieste di I/O che il task intende inviare. Una volta che l'unità è stata aperta, il task può inviare i comandi previsti dal dispositivo impiegando la funzione BeginIO, DoIO o SendIO. Quando il task non ha più necessità d'interagire con il dispositivo Console, dovrebbe chiuderlo (anche se non è strettamente necessario) al fine di liberare memoria occupata inutilmente.

RawKeyConvert

Sintassi di chiamata della funzione

```
numChars = RawKeyConvert (inputEvent, bufferPointer, bufferLength, keyMap)
D0                       A0           A1            D1            A2
```

Scopo della funzione

Questa funzione converte (ossia decodifica) i codici grezzi di un evento di tipo IECLASS_RAWKEY applicando lo standard ANSI X3.64. La conversione è basata sulla struttura KeyMap specificata come argomento della funzione. Per chiamare RawKeyConvert, al pari di CDInputHandler, non occorre aprire alcuna unità del dispositivo. È sufficiente chiamare OpenDevice indicando come argomento unit il valore −1, cosicché venga restituito solo l'indirizzo della struttura Device di gestione del dispositivo. RawKeyConvert necessita di questo indirizzo per interagire con le routine interne del dispositivo Console. In

particolare, si ricordi che l'indirizzo della struttura Device costituisce anche l'indirizzo base della libreria di routine del dispositivo, ed è l'unico riferimento assoluto all'interno della stessa. Quanto detto vale anche per la funzione CDInputHandler.

L'esito della funzione RawKeyConvert viene restituito nella variabile numChars di tipo short: se la trasformazione dei codici grezzi in codici ANSI X3.64 ha avuto successo, numChars contiene il numero effettivo di caratteri ANSI ottenuti con la trasformazione e memorizzati nel buffer indicato dal task (l'indirizzo di questo buffer si indica nell'argomento bufferPointer, e la sua lunghezza nell'argomento bufferLength). Se le dimensioni del buffer dovessero risultare insufficienti per contenere tutti i caratteri ANSI, nella variabile numChars la funzione restituisce il valore −1, per indicare che si è verificata una condizione di overflow. In questo caso, non si può essere certi che tutti i caratteri ANSI presenti nel buffer siano validi; il task deve incrementare le dimensioni del buffer e chiamare nuovamente RawKeyConvert.

Argomenti della funzione

inputEvent — Deve contenere l'indirizzo della struttura InputEvent che definisce l'evento di tipo IECLASS_RAWKEY da trasformare in codici ANSI.

bufferPointer — Deve contenere l'indirizzo del buffer che il task ha allocato per ricevere la traduzione dell'evento allo stato grezzo in codici ANSI.

bufferLength — Deve indicare la dimensione del buffer allocato dal task.

keyMap — Deve contenere l'indirizzo della struttura KeyMap che si desidera venga impiegata per convertire i codici grezzi dei tasti in caratteri ANSI; se l'indirizzo è zero, viene utilizzata la struttura KeyMap di default.

Preparazione della struttura IOStdReq

La funzione RawKeyConvert non richiede come argomento l'indirizzo di una struttura di I/O.

Discussione

RawKeyConvert utilizza una struttura KeyMap per convertire i codici grezzi dei tasti in caratteri standard ANSI X3.64. La struttura KeyMap può essere quella che rappresenta la mappa di default della tastiera, oppure quella indicata dal task nell'argomento keyMap della funzione.

I caratteri ANSI ottenuti dalla conversione vengono memorizzati dalla funzione nel buffer indicato dal task; quando il task riottiene il controllo può procedere alla loro elaborazione. Per il buffer è opportuno specificare sempre dimensioni maggiori di quelle previste per evitare il pericolo dell'overflow.

Le funzioni RawKeyConvert e CDInputHandler effettuano un accesso diretto alle routine interne del dispositivo Console, a differenza delle funzioni BeginIO, DoIO o SendIO.

COMANDI STANDARD DEL DISPOSITIVO

CMD_CLEAR

Scopo del comando

Il comando CMD_CLEAR azzera il buffer interno di lettura del dispositivo Console, che viene impiegato soltanto con il comando CMD_READ. Azzerando il buffer interno prima d'inviare un comando CMD_READ, si ha la certezza che non contenga caratteri estranei a quelli ricevuti.

CMD_CLEAR permette il QuickIO, e restituisce una risposta alla reply port del task soltanto se il QuickIO non è stato accolto. L'esito del comando viene restituito nel parametro io_Error della struttura di I/O. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_ABORTED indica che la richiesta di I/O è stata eliminata.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo, e l'indirizzo della struttura ConUnit di gestione dell'unità;

entrambi questi indirizzi vengono restituiti dalla funzione OpenDevice all'apertura dell'unità. Il task deve impostare io_Command con il comando CMD_CLEAR. Infine, deve inizializzare io_Flags a 0, oppure a IOF_QUICK per richiedere il QuickIO, che può essere accolto o meno a seconda delle condizioni in cui si trova il sistema nel momento in cui viene inviato il comando CMD_CLEAR.

Discussione

Se un task esegue una serie di comandi CMD_READ e desidera assicurarsi prima di ogni invio che il buffer di lettura interno del dispositivo Console sia vuoto, deve far precedere ogni comando CMD_READ da un comando CMD_CLEAR, in modo da azzerare tutti i byte del buffer interno e inizializzarne nuovamente il puntatore. In questo modo il task non corre il rischio di leggere caratteri estranei lasciati nel buffer da precedenti operazioni di lettura.

CMD_CLEAR può essere eseguito sia con il QuickIO sia con il QueuedIO. Se viene accodato, deve attendere che vengano eseguiti tutti i comandi CMD_READ che lo precedono in coda.

CMD_READ

Scopo del comando

Un task chiama il comando CMD_READ per leggere gli eventi di input provenienti dalla tastiera. Ciascun task riceve soltanto gli eventi di tastiera che avvengono quando è selezionata la finestra associata all'unità. Il numero di eventi che si riescono a ottenere con un solo comando CMD_READ dipende dalla rapidità con cui si susseguono. Se per esempio l'utente preme diversi tasti contemporaneamente, gli eventi relativi si susseguono abbastanza in fretta da essere conglobati nello stesso flusso di eventi e nel buffer vengono riportati tutti i loro codici. Un altro caso che può produrre l'accumulo di diversi eventi nella stessa stringa è quello in cui il task, dopo aver aperto una finestra di Intuition, apre un'unità del dispositivo Console e compie varie operazioni prima di controllare con CMD_READ se l'utente ha premuto dei tasti mentre era selezionata quella finestra. Se l'utente preme un tasto nel periodo che intercorre fra l'apertura dell'unità e l'invio del comando CMD_READ, i relativi dati vengono automaticamente memorizzati nel buffer interno di lettura dell'unità (in questo caso nella finestra non appare niente, dal momento che è compito del task generare l'eco). Quando CMD_READ viene finalmente inviato, il task riceve nel suo buffer tutti gli eventi di tastiera che si sono verificati nel frattempo.

Il risultato della lettura eseguita da CMD_READ è sempre una stringa di testo che viene memorizzata nel buffer. La lunghezza di questa stringa viene memorizzata nel parametro io_Actual della struttura di I/O. La stringa può contenere la trasformazione in caratteri ANSI dei codici grezzi oppure gli stessi codici grezzi (rappresentati secondo una particolare sintassi), a seconda che sia abilitata o meno la trasformazione nello standard ANSI X3.64. Per default la trasformazione nello standard ANSI degli eventi grezzi da tastiera è abilitata, ma il task (e lo stesso utente, tramite la tastiera) possono intervenire e disabilitarla con una particolare sequenza escape denominata SRE (set raw events); per riabilitarla la sequenza escape è invece RRE (reset raw events). Nella descrizione del comando CMD_WRITE illustreremo meglio l'uso delle sequenze escape.

Se la trasformazione in ANSI è abilitata, i codici grezzi dei caratteri vengono convertiti tramite la mappa di tastiera, sulla quale si può intervenire tramite i comandi CD_ASKKEYMAP e CD_SETKEYMAP.

CMD_READ consente il QuickIO e restituisce una risposta alla reply port del task soltanto se il QuickIO non è stato accettato. I risultati prodotti dall'esecuzione vengono restituiti nei parametri io_Actual e io_Error. Il parametro io_Actual indica il numero di caratteri effettivamente letti. Il valore 0 di io_Error indica che il comando è stato eseguito. IOERR_ABORTED indica che la richiesta di I/O è stata eliminata. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_BADLENGTH indica che nel parametro io_Length è stato memorizzato un valore non accettabile.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura ConUnit di gestione dell'unità; entrambi questi indirizzi vengono restituiti dalla funzione OpenDevice all'apertura dell'unità. In particolare, la struttura ConUnit individua la finestra dalla quale il task desidera ricevere i dati. Deve inizializzare io_Flags a 0, oppure a IOF_QUICK per richiedere il QuickIO, che può essere o meno accordato a seconda delle condizioni in cui si trova il sistema nel momento in cui viene inviato il comando. Deve inizializzare io_Command a CMD_READ, e inoltre deve inizializzare i seguenti parametri:

- io_Length. Deve contenere il numero di caratteri che si desidera vengano letti dalla tastiera. Può variare fra 1 e la massima dimensione del buffer allocato per ricevere i caratteri. Il valore che si memorizza in io_Length dipende dal tipo di algoritmo impiegato dal task per elaborare i caratteri ricevuti; può per esempio essere utile indicare nel parametro io_Length il valore 1 per elaborare un carattere alla volta. Comunque, a prescindere dalla quantità di dati che di volta in volta

vengono trasferiti nel buffer del task, il buffer interno di lettura dell'unità (il buffer di transito) assicura che nessun evento da tastiera venga perso. Leggendo un carattere alla volta, il task non fa altro che ottenere i caratteri che potrebbe leggere in un colpo solo indicando un valore più grande in io_Length. Si noti infine che io_Length stabilisce il *massimo* numero di caratteri che possono essere letti con un comando CMD_READ, e non il minimo. Quindi, se il numero di eventi di input nel flusso non raggiunge il limite indicato da io_Length, CMD_READ restituisce ugualmente una risposta. Si ricordi inoltre che se gli eventi di input relativi alla tastiera non vengono trasformati in codici ANSI, se cioè giungono nel buffer del task allo stato grezzo, premere un tasto (e anche rilasciarlo) genera una stringa di caratteri organizzati secondo una particolare sintassi che descrive l'evento allo stato grezzo. Se il task ha indicato in io_Length un valore sufficientemente grande, riceve nel proprio buffer l'intera stringa. Vediamo un esempio del tipo di stringa che si riceve quando è stata disabilitata la trasformazione in codici ANSI dei codici grezzi da tastiera tramite un'opportuna sequenza SRE. Ricordiamo che questa sequenza prevede la sintassi <CSI>Tipo1;Tipo2;...{, dove Tipo1, Tipo2... devono essere numeri compresi fra 0 e 16; questi numeri disattivano la generazione dei codici ANSI per 17 tipi di eventi (tastiera, mouse, cursore, gadget...). Nel nostro esempio, desideriamo disabilitare la generazione dei codici ANSI relativa agli eventi da tastiera, e quindi la sequenza SRE è: **<CSI>1{**. Da tastiera può essere impartita digitando in sequenza i tasti ESC, [, 1, {. Se ora l'utente preme la barra spaziatrice, il task ottiene nel proprio buffer la seguente stringa di testo:

<CSI>1;0;64;32768;16384;16384;342910772;39349

Questa stringa di testo, tramite una sintassi che non analizzeremo in questo volume, descrive l'evento grezzo relativo alla pressione della barra spaziatrice. Se dopo un tempo inferiore al limite di ripetizione degli eventi da tastiera l'utente rilascia il tasto, il task ottiene, se ha inviato un altro comando CMD_READ, la seguente stringa di testo:

<CSI>1;0;192;32768;0;0;342910772;379355

Si noti che si tratta di due stringhe di testo, cioè di stringhe composte da caratteri ASCII, e che il simbolo <CSI> è il Control Sequence Introducer, un byte che vale 0x9B. Quando è l'utente a impartire una sequenza escape da tastiera, il CSI si ottiene premendo in sequenza i tasti ESC e [. Se la traduzione in ANSI è attiva, tale sequenza viene trasformata nel codice 0x9B.

Un altro esempio di input di eventi allo stato grezzo si ottiene se nella sequenza SRE al posto del Tipo1 specifichiamo il Tipo2, cioè se impartiamo da tastiera i caratteri ESC, [, 2, {. In questo modo gli eventi di input relativi alle pressioni dei tasti vengono tradotti in codici ANSI, ma se l'utente preme all'interno della finestra il pulsante sinistro del

mouse, il task ottiene una stringa di testo simile alla seguente:

<CSI>2;0;104;49152;0;0;342911349;8397

Quando poi l'utente rilascia il pulsante, al task perviene una stringa di testo come la seguente:

<CSI>2;0;232;32768;0;0;342911349;828420

Infine, si ricordi che per ripristinare la traduzione in ANSI per una o più categorie di eventi, occorre inviare al dispositivo la sequenza RRE, che mantiene la stessa sintassi di SRE salvo per la parentesi graffa, che dev'essere di chiusura.

- io_Data. Si deve inizializzare questo parametro con l'indirizzo del buffer di lettura del task (buffer di input), nel quale il comando CMD_READ trasferisce i caratteri nella quantità indicata da io_Length.

Discussione

CMD_READ permette a un task di ricevere in un buffer da lui definito una serie di eventi di input provenienti dalla tastiera e trasferiti nel buffer interno dell'unità. Generalmente si ricevono caratteri ANSI X3.64, cioè eventi di input grezzi trasformati, ma all'occorrenza un task può ricevere gli eventi allo stato grezzo.

Di solito il comando CMD_READ viene definito in modo da richiedere un carattere per volta. Tuttavia, se il parametro io_Length della struttura IOStdReq viene inizializzato con un valore maggiore di 1, CMD_READ legge diversi caratteri in successione (purché gli eventi si susseguano con sufficiente rapidità, oppure nel caso che sia stata disabilitata la trasformazione in codici ANSI). Il valore restituito dal comando nel parametro io_Actual è il numero di caratteri effettivamente letti, e può essere copiato direttamente nel parametro io_Length di una richiesta di scrittura (CMD_WRITE) per generare prontamente l'eco dei caratteri digitati; in questo caso, nel parametro io_Data della richiesta di scrittura dev'essere presente l'indirizzo dello stesso buffer impiegato in lettura.

Tutti i tasti che dispongono di un equivalente ASCII vengono tradotti dalle routine interne del dispositivo Console utilizzando la mappa di tastiera di default, o quella definita dal comando CD_SETKEYMAP. Per i tasti che non dispongono dell'equivalente in ASCII, il dispositivo genera una sequenza escape e la inserisce nel flusso di input del task. Nel modo di default, per esempio, i tasti funzione da F1 a F10 e i tasti di spostamento del cursore provocano l'inserimento nel flusso di sequenze escape.

Se viene inviato un comando CMD_READ e nel buffer di lettura dell'unità non è presente alcun evento di input (l'utente non ha premuto nessun tasto), il comando non restituisce una risposta fino a quando non arriva almeno un

evento. Se invece si verificano eventi in successione a una rapidità sufficiente per essere conglobati tutti nello stesso flusso, ma la stringa che CMD_READ trasferisce dal buffer dell'unità a quello del task è più corta di quanto previsto dal parametro io_Length, il comando restituisce ugualmente una risposta, anche se il valore inserito nel parametro io_Actual sarà diverso da quello presente in io_Length. Il valore che il task indica in io_Length stabilisce infatti un limite superiore e non inferiore.

CMD_WRITE

Scopo del comando

Il comando CMD_WRITE provoca la visualizzazione di una stringa di caratteri nella finestra di Intuition associata con l'unità del dispositivo Console. CMD_WRITE visualizza i caratteri contenuti nel buffer di scrittura del task, trattandoli come caratteri in standard ANSI X3.64. Il numero di caratteri da visualizzare è indicato dal task nel parametro io_Length. Se però il task vi inserisce il valore −1, CMD_WRITE visualizza la stringa fino a quando non incontra il carattere NULL.

I caratteri possono essere ASCII (dal codice 0x20 al codice 0x7E, e dal codice 0xA0 al codice 0xFF), o costituire comandi di controllo dello schermo. Intuition, tramite la struttura Window che definisce la finestra, controlla automaticamente quante righe e quanti caratteri per riga possono essere visualizzati nella finestra. Intuition controlla anche il wordwrap dei testi (la procedura che manda a capo una parola quando una lettera uscirebbe dal margine destro).

La maggior parte dei caratteri di controllo dello schermo (per esempio, il Return o il Backspace) vengono tradotti negli equivalenti caratteri ANSI. Si veda l'*Amiga ROM Kernel Reference Manual* per informazioni sui caratteri di controllo dello schermo.

Ogni volta che viene inviato un comando CMD_WRITE, Intuition prende in considerazione i valori contenuti nei parametri della struttura ConUnit che corrispondono a quelli di una struttura RastPort. Quindi, se il task desidera per esempio visualizzare un testo modificando il colore del tratto, deve alterare il parametro cu_FgPen della struttura ConUnit prima d'inviare il comando CMD_WRITE. Si noti inoltre, che variando il puntatore cu_Window della struttura ConUnit si ridirige in pratica l'output verso una particolare finestra. Questa operazione può essere compiuta prima di qualsiasi comando CMD_WRITE o CMD_READ.

Se per un comando CMD_WRITE viene richiesto il QuickIO senza successo, il comando viene trattato come una richiesta di I/O accodato (QueuedIO) e la relativa struttura viene restituita alla reply port del task. L'esito del comando viene restituito nel parametro io_Error della struttura di I/O. Il valore 0 indica

che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_ABORTED indica che la richiesta di I/O è stata eliminata. IOERR_BADLENGTH indica che il parametro io_Length è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura ConUnit di gestione dell'unità; entrambi questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando CMD_WRITE. Inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che può essere accolto o meno a seconda delle condizioni in cui si trova il sistema nel momento in cui viene inviato il comando CMD_WRITE). Si devono inoltre inizializzare i seguenti parametri:

- io_Length. Si deve inizializzare questo parametro con il numero di caratteri che si vogliono visualizzare nella finestra associata all'unità; se il task indica il valore −1, la visualizzazione continua fino al primo carattere NULL.

- io_Data. Si deve inizializzare questo parametro con l'indirizzo del buffer di scrittura del task, nel quale si trova la stringa di testo da visualizzare.

Discussione

CMD_WRITE permette a un task d'inviare testi e codici di controllo a una finestra di Intuition associata con l'unità del dispositivo Console che ha aperto. Il task deve preparare un buffer contenente tutti i caratteri che desidera inviare alla finestra. Questo è il modo in cui un task invia i caratteri a una finestra di Intuition e ne controlla l'impaginazione.

L'associazione della finestra all'unità viene attuata al momento della chiamata a OpenDevice. Il comando CMD_WRITE può essere inviato a qualsiasi unità del dispositivo Console (ognuna associata a una diversa finestra di Intuition) cambiando semplicemente l'indirizzo della struttura ConUnit nel puntatore io_Unit e riutilizzando la stessa struttura di I/O. Se invece si desidera inviare contemporaneamente più comandi a più unità, occorre impiegare una struttura di I/O diversa per ogni invio.

I comandi di controllo dello schermo servono per formattare i testi durante la loro visualizzazione e per variare gli stili con cui i caratteri vengono rappresentati. Questi comandi si suddividono in due classi.

- Quelli costituiti da un singolo byte, fra cui il carattere line feed e il carattere di tabulazione orizzontale (in tutto sono otto, e la loro descrizione si può trovare nell'*Amiga ROM Kernel Reference Manual*).

- Quelli costituiti da una sequenza escape, cioè da una serie di caratteri conformi allo standard ANSI X3.64 e alle specifiche previste dall'Amiga. Una sequenza escape deve sempre iniziare con il carattere CSI (Control Sequence Introducer; il suo codice è 0x9B), il quale viene poi seguito da un certo numero di caratteri che variano a seconda del comando. Una sequenza escape potrebbe essere "<CSI>5;5H", che muove il cursore alla posizione (5,5) sullo schermo. Le sequenze escape previste dallo standard ANSI X3.64 e dall'Amiga sono molte: agiscono sul cursore, sul testo (cancellazione, inserimento di righe, e così via); determinano il formato della pagina (altezza della pagina, lunghezza delle righe e così via). Altre sequenze escape sono i comandi SRE (set raw events) e RRE (reset raw events), e il comando per leggere lo stato della finestra, grazie al quale il task può avere informazioni sulla pagina di testo che occupa la finestra (posizione della riga superiore, della riga inferiore, della colonna di sinistra e di destra). Parlando del comando CMD_READ abbiamo spiegato che l'utente può digitare una sequenza escape sulla tastiera e ottenere, per esempio, la disabilitazione del filtro che traduce i codici grezzi di tastiera in caratteri ANSI. Ora vediamo invece come il task può inviare le sequenze escape all'unità del dispositivo. Supponiamo che il task desideri visualizzare una frase in corsivo e al termine desideri disabilitare la trasformazione ANSI dei codici grezzi. Se la frase è "Invio la sequenza escape SRE per gli eventi di tastiera.", all'interno del sorgente in C del task il programmatore deve inizializzare i parametri io_Data e io_Length nel modo sequente.

```
iOStdReq->io_Data = (APTR)"\x9b3minvio la sequenza escape SRE
                          per gli eventi di tastiera.\x9b1{";
iOStdReq->io_Length = -1L;
```

La prima sequenza escape abilita il corsivo, mentre la seconda corrisponde alla sequenza escape SRE, che disabilita il filtro ANSI. Se ora il task invia un comando CMD_READ, riceve i successivi eventi allo stato grezzo. Nel corso della descrizione del comando CMD_READ sono riportati esempi di questo tipo di stringhe.

COMANDI SPECIFICI DEL DISPOSITIVO

CD_ASKDEFAULTKEYMAP

Scopo del comando

CD_ASKDEFAULTKEYMAP copia nel buffer indicato dal task i parametri della struttura KeyMap che definisce la mappa di tastiera prevista per default dal dispositivo Console. La stessa mappa viene utilizzata come mappa di default anche dalla funzione RawKeyConvert, qualora nella chiamata venga specificato un puntatore alla struttura KeyMap nullo. CD_ASKDEFAULTKEYMAP permette il QuickIO e restituisce una risposta alla reply port del task solo se il QuickIO non viene accolto.

L'esito del comando viene restituito nel parametro io_Error della struttura di I/O. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_ABORTED indica che la richiesta di I/O è stata eliminata, e IOERR_BADLENGTH indica che il parametro io_Length è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo, e l'indirizzo della struttura ConUnit di gestione dell'unità; entrambi questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando CD_ASKDEFAULTKEYMAP. Inizializzare io_Flags a 0, oppure a IOF_QUICK per richiedere il QuickIO (che può essere accolto o meno a seconda delle condizioni in cui si trova il sistema nel momento in cui viene inviato il comando). Inoltre, deve inizializzare i seguenti parametri:

- io_Data. Si deve inizializzare questo parametro con l'indirizzo del buffer allocato dal task che contiene la copia della struttura KeyMap di default della tastiera.

- io_Length. Si deve inizializzare questo parametro con il numero di byte che compongono la struttura KeyMap; a questo scopo, un task può utilizzare l'operatore sizeof() del linguaggio C.

Discussione

Il comando CD_ASKDEFAULTKEYMAP copia nel buffer indicato dal task la struttura KeyMap che definisce la mappa di default della tastiera. Serve quando si desidera ripristinare la mappa di default per una particolare unità.

CD_ASKKEYMAP

Scopo del comando

CD_ASKKEYMAP copia nel buffer del task i parametri della struttura KeyMap che definisce la mappa di tastiera associata all'unità dalla struttura di I/O. CD_ASKKEYMAP permette il QuickIO e restituisce una risposta alla reply port del task solo se il QuickIO non viene accolto.

L'esito del comando viene restituito nel parametro io_Error della struttura di I/O. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_ABORTED indica che la richiesta di I/O è stata eliminata. e IOERR_BADLENGTH indica che il parametro io_Length è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo, e l'indirizzo della struttura ConUnit di gestione dell'unità; entrambi questi indirizzi vengono restituiti dalla funzione OpenDevice all'apertura dell'unità. Il task deve impostare io_Command con il comando CD_ASKKEYMAP. Inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che può essere accolto o meno a seconda delle condizioni in cui si trova il sistema nel momento in cui viene inviato il comando). Inoltre, deve inizializzare i seguenti parametri:

- io_Data. Deve contenere l'indirizzo del buffer allocato dal task, in cui va copiata la struttura KeyMap che definisce la mappa di tastiera associata all'unità.

- io_Length. Deve contenere il numero di byte che compongono la struttura KeyMap; a questo scopo, un task può utilizzare l'operatore sizeof() del linguaggio C.

Discussione

Il comando CD_ASKKEYMAP copia nel buffer indicato dal task la struttura KeyMap che definisce la mappa di tastiera associata all'unità.

CD_SETDEFAULTKEYMAP

Scopo del comando

Questo comando serve per rendere di default la mappa di tastiera definita dal task con un'opportuna struttura KeyMap. Dopo l'esecuzione del comando, la nuova mappa viene impiegata per default ogni volta che viene aperta un'unità del dispositivo. CD_SETDEFAULTKEYMAP permette il QuickIO e restituisce una risposta alla reply port del task solo se il QuickIO non è stato accolto.

L'esito del comando viene restituito nel parametro io_Error della struttura di I/O. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_ABORTED indica che la richiesta di I/O è stata eliminata e IOERR_BADLENGTH indica che il parametro io_Length è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo, e l'indirizzo della struttura ConUnit di gestione dell'unità; entrambi questi indirizzi vengono restituiti dalla funzione OpenDevice all'apertura dell'unità. Il task deve impostare io_Command con il comando CD_SETDEFAULTKEYMAP. Deve inoltre inizializzare io_Flags a 0, oppure a IOF_QUICK per richiedere il QuickIO (che può essere accolto o meno a seconda delle condizioni in cui si trova il sistema nel momento in cui viene inviato il comando). Inoltre, deve inizializzare i seguenti parametri:

- io_Data. Si deve inizializzare questo parametro con l'indirizzo della struttura KeyMap allocata e inizializzata dal task.

- io_Length. Si deve inizializzare questo parametro con il numero di byte che compongono la struttura KeyMap; a questo scopo, un task può utilizzare l'operatore sizeof() del linguaggio C.

Discussione

Il comando CD_SETDEFAULTKEYMAP copia nella struttura KeyMap che rappresenta la mappa di default della tastiera il contenuto della struttura KeyMap indicata dal task. Il task può così variare in modo permanente e globale la mappa di tastiera. Si ricordi che la mappa di tastiera associata a un'unità non viene impiegata se il task ha disabilitato la trasformazione dei codici grezzi in caratteri ANSI.

CD_SETKEYMAP

Scopo del comando

CD_SETKEYMAP serve per associare all'unità del dispositivo una particolare mappa di tastiera definita dal task all'interno di una struttura KeyMap. CD_SETKEYMAP permette il QuickIO e restituisce una risposta alla reply port del task solo se il QuickIO non è stato accolto.

L'esito del comando viene restituito nel parametro io_Error della struttura di I/O. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_ABORTED indica che la richiesta di I/O è stata eliminata, e IOERR_BADLENGTH indica che il parametro io_Length è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura ConUnit di gestione dell'unità; entrambi questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando CD_SETKEYMAP. Inizializzare io_Flags a 0, oppure a IOF_QUICK per richiedere il QuickIO (che può essere accolto o meno a seconda delle condizioni in cui si trova il sistema nel momento in cui viene inviato il comando). Inoltre, deve inizializzare i seguenti parametri:

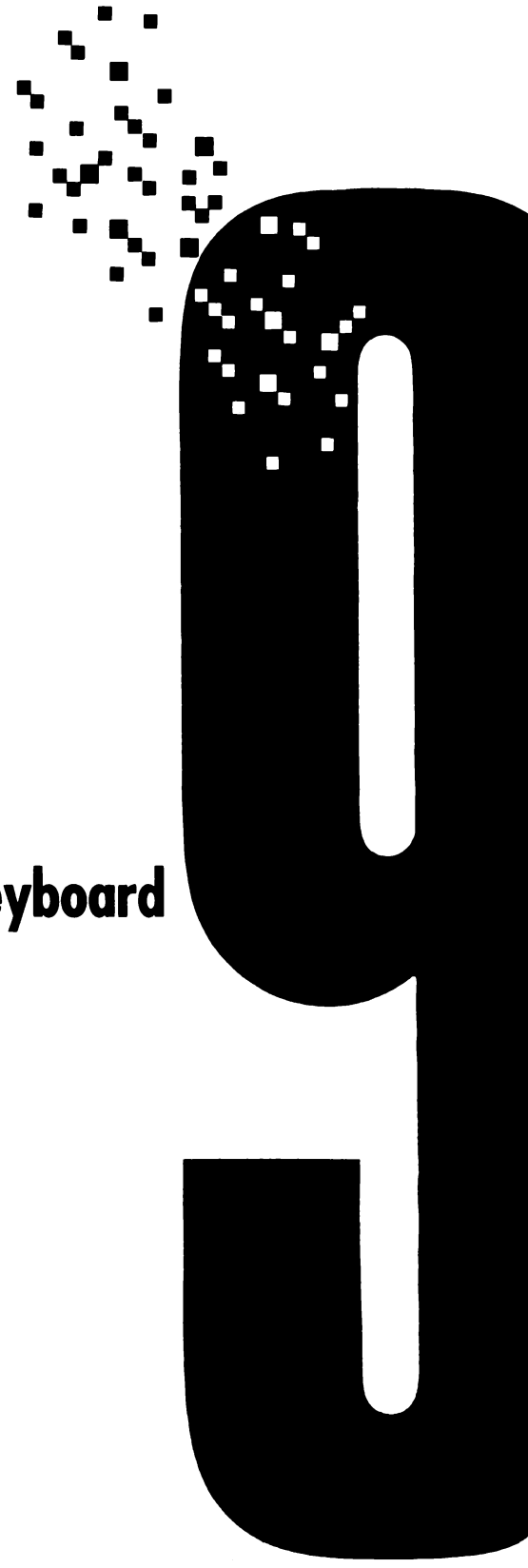
- io_Data. Si deve inizializzare questo parametro con l'indirizzo della struttura KeyMap allocata e inizializzata dal task.

- io_Length. Si deve inizializzare questo parametro con il numero di byte

che compongono la struttura KeyMap; a questo scopo, un task può utilizzare l'operatore sizeof( ) del linguaggio C.

Discussione

Il comando CD_SETKEYMAP copia all'interno della struttura cu_KeyMapStruct (sotto-struttura della struttura ConUnit associata all'unità) il contenuto della struttura KeyMap indicata dal task nella richiesta di I/O. In questo modo il task riesce a modificare la mappa di tastiera impiegata da una particolare unità al fine di personalizzarla.



Il dispositivo Keyboard

Introduzione

Il dispositivo Keyboard è responsabile dell'acquisizione dei dati immessi dall'utente attraverso la tastiera. Opera in accesso condiviso e possiede una sola unità, l'unità 0. Nell'Amiga 2000 e nell'Amiga 500 questo dispositivo risiede su ROM; nell'Amiga 1000 viene automaticamente caricato dal disco del Kickstart nella WCS ROM e pertanto si può considerare anche in questo caso residente su ROM.

Un evento da tastiera viene descritto tramite una struttura InputEvent che contiene diverse informazioni: il codice grezzo relativo al tasto premuto, la distinzione tra pressione e rilascio del tasto, lo stato dei tasti qualificatori, la distinzione fra tastierina numerica e il resto della tastiera. Il dispositivo Keyboard ottiene dall'hardware queste informazioni e le organizza all'interno di una struttura InputEvent, creando così un evento da tastiera (la struttura InputEvent è stata illustrata nel capitolo 7).

Il dispositivo Keyboard "ricorda" quali tasti vengono premuti inserendo per ognuno di essi una struttura InputEvent nel suo buffer di tastiera. Questo buffer viene gestito automaticamente dalle routine interne del dispositivo Keyboard, e non dev'essere confuso con il buffer che il task definisce e impiega con il comando KBD_READEVENT.

Una volta che gli eventi di input si trovano nel buffer di tastiera del dispositivo Keyboard, possono essere elaborati in diversi modi. L'AmigaDOS, per esempio, elabora gli eventi che inducono al reset del sistema, mentre un task può elaborare altri eventi di input secondo quanto imposto dal programmatore. Tuttavia, se né l'AmigaDOS né un task elaborano gli eventi, è il dispositivo Input che li riceve, e li congloba con quelli provenienti dai dispositivi Gameport, TrackDisk, e da ogni altra sorgente di input del sistema Amiga, creando quello che è stato definito "flusso degli eventi di input".

Il dispositivo Keyboard permette a un task, tramite il comando KBD_READMATRIX, di rilevare lo stato (premuto o rilasciato) di ogni tasto della tastiera. Oltre a questo comando, il dispositivo Keyboard prevede anche KBD_ADDRESETHANDLER, KBD_REMRESETHANDLER e KBD_RESETHANDLERDONE, tramite i quali i task possono aggiungere funzioni di gestione del reset alla procedura standard prevista dal sistema. Queste funzioni vengono inserite nella lista di reset mantenuta dal sistema e vengono eseguite prima della procedura di reset standard, nella sequenza definita dai loro rispettivi livelli di priorità. Con questi comandi i task possono organizzare una serie di routine che intervengono quando l'utente digita la combinazione di reset Ctrl/Amiga-Sinistro/Amiga-Destro, per salvaguardare ad esempio particolari dati che andrebbero inevitabilmente perduti.

Funzionamento del dispositivo Keyboard

La Figura 9.1 illustra il funzionamento generale del dispositivo Keyboard. Il dato che ha origine nella tastiera (pressione o rilascio di un tasto), viene inserito nel buffer interno del dispositivo Keyboard (tramite la struttura InputEvent che lo rappresenta) e se non viene intercettato da un comando KBD_READEVENT viene infine ceduto al dispositivo Input per ulteriori elaborazioni.

Come si può osservare in figura, un task può creare diverse funzioni di gestione del reset di sistema provocato dall'utente attraverso la tastiera. Queste funzioni vengono ordinate nella lista di reset secondo le priorità indicate dai rispettivi parametri ln_Pri: sono le prime procedure che vengono eseguite quando l'utente imposta la combinazione di tasti che provoca il reset. Successivamente, quando tutte le funzioni della lista sono state eseguite, l'Amiga esegue la propria procedura di reset standard, che è predefinita nel software sistema.

Come si nota dalla Figura 9.1, grazie al buffer di tastiera mantenuto dal dispositivo Keyboard l'utente può digitare a qualunque velocità senza che venga perso alcun dato, anche quando il sistema è momentaneamente occupato in altre attività.

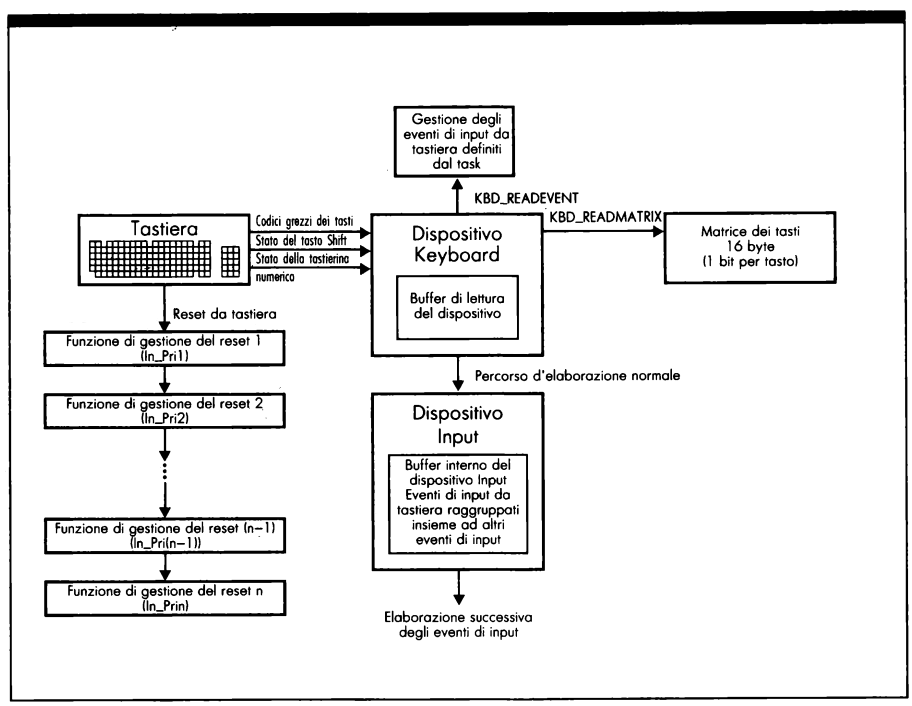


Figura 9.1: *Funzionamento del dispositivo Keyboard*

Elaborazione degli eventi di input da tastiera

Tutti gli eventi di input da tastiera vengono elaborati da una sequenza predefinita di routine, come si può vedere nella Figura 9.2 (nella pagina successiva). Gli eventi hanno origine nell'hardware della tastiera sotto forma di segnali elettrici, e successivamente vengono analizzati dalle routine interne del dispositivo Keyboard per essere elaborati.

Sulla base dei segnali provenienti dall'hardware, le routine interne del dispositivo trasformano gli eventi in strutture InputEvent che vengono inserite nel buffer interno del dispositivo Keyboard. L'elaborazione degli eventi si svolge quindi nella sequenza sotto riportata.

1. Se l'AmigaDOS è attivo, le sue routine interne leggono il contenuto del buffer di tastiera e filtrano (cioè, elaborano) gli eventi di input per i quali sono state programmate. In particolare, le routine dell'AmigaDOS intercettano sempre la sequenza di reset da tastiera dell'Amiga (escludendo ovviamente il caso in cui un programma prende il controllo della macchina, cioè disabilita completamente la gestione multi-tasking).

2. Gli input da tastiera che non vengono intercettati dalle routine dell'AmigaDOS, rimangono disponibili per ulteriori elaborazioni. A questo punto, un task può memorizzarli nel proprio buffer di lettura utilizzando il comando KBD_READEVENT. Questo procedimento permette al task di elaborare la pressione di un tasto senza passare attraverso il dispositivo Input o il dispositivo Console.

3. Gli eventi di input da tastiera che non vengono intercettati né dall'AmigaDOS né da un task, giungono alle routine interne del dispositivo Input per ulteriori elaborazioni. Il dispositivo Input riunisce in un unico flusso gli eventi in arrivo dai dispositivi Keyboard, Gameport e TrackDisk, e gli eventi di input definiti dai task (si veda il capitolo 7). Le routine interne del dispositivo Input cedono poi il flusso di eventi alla catena delle funzioni di gestione degli eventi di input.

comandi del dispositivo Keyboard

Per programmare il dispositivo Keyboard si utilizzano cinque comandi specifici e due comandi standard. Alcuni sono eseguibili con il QuickIO, ma nessuno prevede il modo immediato. Tutti i comandi influenzano il parametro io_Error della struttura IOStdReq; CMD_CLEAR influenza anche il contenuto del buffer di tastiera del dispositivo Keyboard.

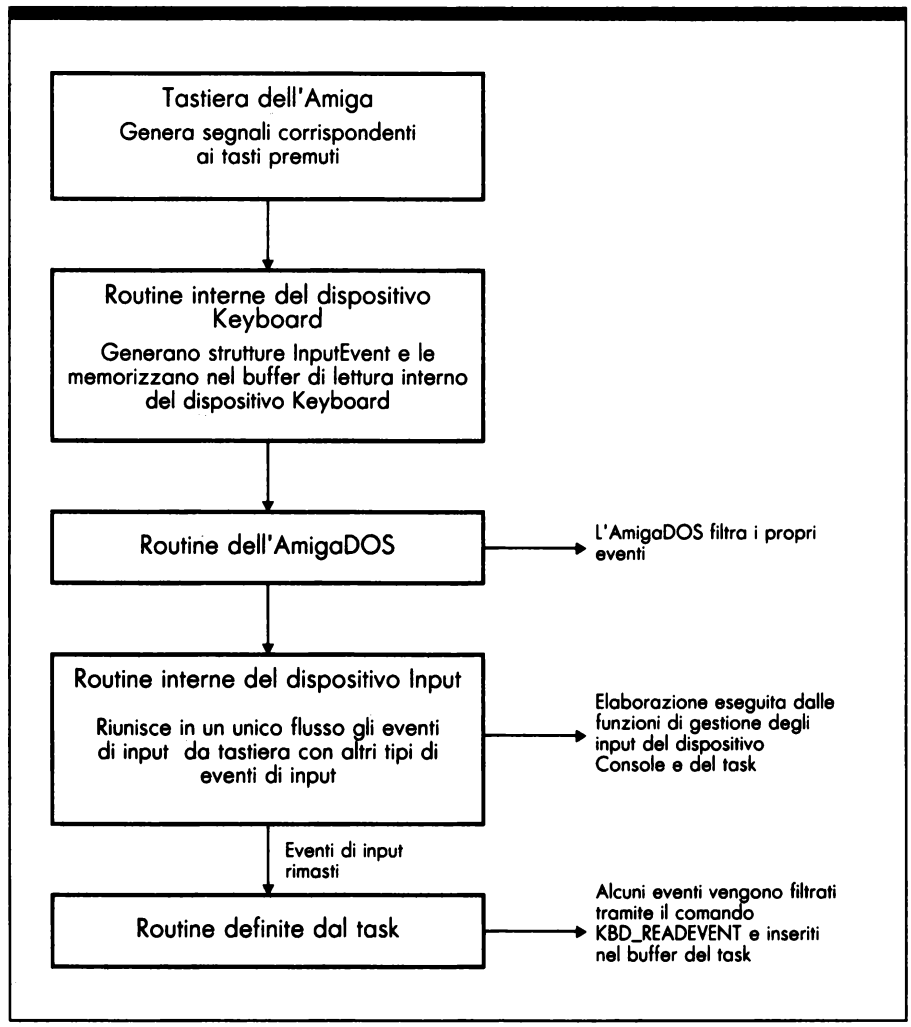


Figura 9.2: *Elaborazione degli eventi di input da tastiera eseguita dal dispositivo Keyboard*

L'invio dei comandi al dispositivo Keyboard

La Figura 9.3 (nella pagina successiva) mostra lo schema generale utilizzato per inviare comandi alle routine interne del dispositivo. Le linee con le frecce rappresentano i parametri da inizializzare e quelli che vengono restituiti dalle routine interne del dispositivo.

L'interazione con Keyboard, dopo l'apertura, prevede tre fasi.

1. *Preparazione della struttura di I/O*. In questa fase il programmatore ha un controllo completo. Qui vengono inizializzati i parametri della struttura IOStdReq, perché le routine interne del dispositivo Keyboard

possano elaborare un comando. Il comando KBD_ADDRESETHANDLER richiede che i parametri is_Data e is_Code di una struttura Interrupt rappresentino una funzione di gestione del reset di sistema. I parametri da inizializzare dipendono dal comando che si desidera inviare al dispositivo Keyboard.

2. *Invio ed elaborazione della richiesta*. L'unico compito del programmatore in questa fase è l'invio del comando al dispositivo tramite le funzioni BeginIO, DoIO o SendIO. Il controllo passa alle routine interne del dispositivo, che procedono a elaborare la richiesta.

3. *Elaborazione e restituzione dei parametri*. Le routine interne del dispositivo Keyboard e il sistema hanno un completo controllo sui valori restituiti nei parametri quando l'elaborazione del comando è stata completata. I risultati prodotti dall'elaborazione del comando vengono restituiti al task che l'aveva originariamente inviato. Se la richiesta non prevede il QuickIO, il comando viene elaborato nel momento in cui raggiunge la sommità della coda alla request port del dispositivo. Il dispositivo Keyboard restituisce quindi la richiesta di I/O nella coda alla reply port del task. Se invece la richiesta prevede il QuickIO (e il QuickIO viene accordato), la risposta non viene accodata alla reply port del task, ma giunge direttamente al task con il flag IOF_QUICK ancora impostato.

Tutti i comandi del dispositivo Keyboard segnalano l'esito del comando restituendo l'opportuno valore nel parametro io_Error della struttura IOStdReq. Inoltre, il comando KBD_READEVENT riempie il buffer del task con le strutture InputEvent che caratterizzano i nuovi eventi di input da tastiera; KBD_READMATRIX riempie il buffer del task con la matrice dei tasti.

La Figura 9.3 mostra anche i parametri che intervengono nell'inizializzazione e nella gestione del dispositivo Keyboard. Le funzioni OpenDevice e

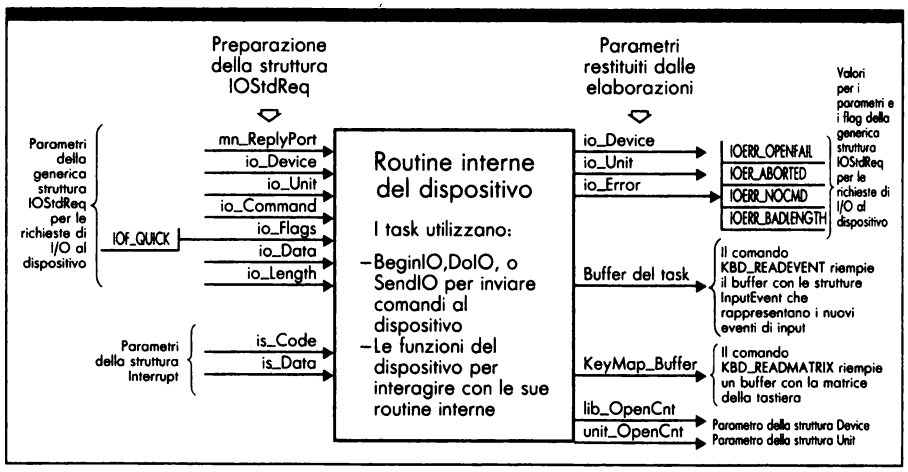


Figura 9.3: *Gestione delle funzioni e dei comandi previsti dal dispositivo Keyboard*

CloseDevice influenzano il parametro lib_OpenCnt della struttura Device; OpenDevice influenza anche il parametro io_Error.

Le strutture del dispositivo Keyboard

Il dispositivo Keyboard non possiede strutture specifiche per l'impiego dei suoi comandi. Tuttavia, le strutture KeyMap, KeyMapNode e KeyMapResource giocano un ruolo importante anche nelle operazioni del dispositivo Keyboard. Per le definizioni delle citate strutture si veda il capitolo 8.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOStdReq)
A1

Scopo della funzione

Questa funzione chiude l'accesso all'unità 0, l'unica prevista dal dispositivo Keyboard. Se il task ha già chiuso i dispositivi Console e Input, anche il dispositivo Keyboard viene automaticamente chiuso.

CloseDevice inizializza a −1 i parametri io_Device e io_Unit appartenenti alla struttura IOStdReq del task. Se in seguito vuole utilizzare ancora l'unità, il task deve inizializzare nuovamente questi parametri tramite una chiamata a OpenDevice. CloseDevice decrementa di 1 il parametro lib_OpenCnt della struttura Device.

Argomenti della funzione

iOStdReq — Deve contenere l'indirizzo della struttura di tipo IOStdReq che il task impiega per interagire con il dispositivo.

Discussione

CloseDevice chiude l'accesso di un task all'unità 0 del dispositivo. Un task dovrebbe sempre verificare che tutte le sue richieste di I/O siano state restituite, prima di chiamare CloseDevice. Per effettuare questo controllo è possibile utilizzare le funzioni GetMsg, Remove, CheckIO e WaitIO.

Si ricordi sempre che il dispositivo Keyboard viene aperto indirettamente al momento dell'apertura dei dispositivi Input o Console; viene inoltre aperto automaticamente dall'AmigaDOS. La procedura adottata per aprirlo determina il modo in cui si deve operare per la sua chiusura.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("keyboard.device", ØL, iOStdReq, ØL)
DØ                 AØ                  DØ A1       D1
```

Scopo della funzione

OpenDevice permette a un task l'accesso all'unità 0, l'unica prevista dal dispositivo Keyboard. Questo dispositivo opera nel modo di accesso condiviso e viene aperto automaticamente dall'AmigaDOS, oppure quando vengono aperti i dispositivi Console e Input.

Una volta che il dispositivo Keyboard è stato aperto, OpenDevice ne inizializza i parametri comuni a tutti i dispositivi. Inoltre, OpenDevice incrementa di 1 il parametro lib_OpenCnt appartenente alla strutture Device di gestione del dispositivo.

OpenDevice richiede una reply port opportunamente inizializzata e un bit di segnalazione del task allocato a quella message port qualora il task desideri essere avvertito nel momento in cui le routine interne del dispositivo Keyboard restituiscono la risposta. I risultati prodotti dall'esecuzione della funzione sono i seguenti:

- io_Device. Individua la struttura Device di gestione del dispositivo. La struttura Device contiene tutte le informazioni necessarie alla gestione del dispositivo Keyboard e tiene il conto del numero di task l'hanno aperto.

- io_Unit. Individua la struttura Unit di gestione dell'unità 0. Questa

struttura contiene la request port dell'unità alla quale giungono le richieste di I/O inviate dai task.

- io_Error. Il valore 0 indica che la richiesta di apertura del dispositivo ha avuto successo. IOERR_OPENFAIL indica che il dispositivo Keyboard non può essere aperto (questo errore può verificarsi, per esempio, nel caso che la memoria disponibile non sia sufficiente). IOERR_NOCMD indica che il parametro io_Command non è stato specificato correttamente.

Argomenti della funzione

"keyboard.device"	Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Keyboard.
0L	Indica che s'intende aprire l'unità 0 del dispositivo, l'unica disponibile.
IOStdReq	Deve contenere l'indirizzo della struttura di tipo IOStdReq che il task intende impiegare per interagire con il dispositivo.
0L	Indica che l'argomento flag non viene preso in considerazione dal dispositivo Keyboard.

Preparazione della struttura IOStdReq

Si deve inizializzare mn_ReplyPort in modo che punti a una struttura MsgPort per la reply port del task. Per allocare e inizializzare una struttura IOStdReq, il task può impiegare la funzione CreateStdIO di supporto alla libreria Exec.

Discussione

La funzione OpenDevice permette a un task di accedere alle routine interne del dispositivo Keyboard. Una caratteristica che rende unico il dispositivo Keyboard è che viene aperto automaticamente dall'AmigaDOS, oppure all'apertura dei dispositivi Input o Console.

Una volta che il dispositivo Keyboard è stato aperto, un task può inviare comandi KBD_READEVENT e altri comandi del dispositivo tramite le funzioni BeginIO, DoIO oppure SendIO.

COMANDI STANDARD DEL DISPOSITIVO

CMD_CLEAR

Scopo del comando

CMD_CLEAR azzera il buffer di tastiera del dispositivo Keyboard. Questo buffer è interno al dispositivo e ha lo scopo di memorizzare le strutture InputEvent relative agli eventi di tastiera a mano a mano che si verificano. Grazie a questo buffer, gli eventi di input corrispondenti ai tasti premuti non vengono "persi" se il sistema è momentaneamente occupato in altre attività. I dati presenti nel buffer di tastiera possono essere letti in qualunque momento attraverso il comando KBD_READEVENT.

Il comando prevede il QuickIO e viene sempre restituito alla reply port del task qualora il QuickIO non sia stato accordato. Il suo esito viene indicato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato, mentre IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità 0; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando CMD_CLEAR, e inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che ha successo o meno a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando).

Discussione

Il dispositivo Keyboard mantiene automaticamente un buffer interno (o di tastiera) per non perdere gli eventi di input che non vengono immediatamente richiesti da un comando KBD_READEVENT. Il comando CMD_CLEAR viene impiegato per pulire il buffer, eliminando quindi tutti gli eventi di input in esso

mantenuti. Se un task, prima di procedere all'invio del comando KBD_READEVENT, vuole assicurarsi che il buffer interno del dispositivo Keyboard non contenga vecchie informazioni, deve preventivamente inviare il comando CMD_CLEAR. In questo modo si evita che il comando KBD_READEVENT restituisca al task informazioni non desiderate. Il dispositivo Keyboard non prevede un buffer interno di scrittura, dal momento che non possiede alcun comando di tipo CMD_WRITE.

Il comando CMD_CLEAR può essere accodato alla request port o elaborato con il QuickIO. Se viene semplicemente accodato, il comando agisce quando raggiunge la sommità della coda e quindi non pulisce immediatamente il buffer del dispositivo. Nel periodo che intercorre fra l'invio del comando e la sua esecuzione, il dispositivo può eseguire un comando KBD_READEVENT che precede CMD_CLEAR nella coda: sugli esiti di questi comandi, CMD_CLEAR non ha alcun effetto.

CMD_RESET

Scopo del comando

CMD_RESET riporta il dispositivo Keyboard allo stato iniziale; tutti i parametri interni e i flag vengono inizializzati ai rispettivi valori di default.

Il comando prevede il QuickIO e la sua struttura viene restituita alla reply port del task soltanto se il QuickIO non viene accolto. L'esito del comando è indicato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato, mentre IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità 0; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando CMD_RESET, e inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che ha successo o meno a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando).

Discussione

CMD_RESET è un comando che ha effetti distruttivi, dal momento che eseguendolo si perdono tutti gli eventi di input che non sono ancora stati elaborati e che si trovano in attesa nel buffer interno del dispositivo.

COMANDI SPECIFICI DEL DISPOSITIVO

KBD_ADDRESETHANDLER

Scopo del comando

Il comando KBD_ADDRESETHANDLER permette di aggiungere una nuova funzione di reset alla lista delle funzioni chiamate durante un reset prima che la macchina venga nuovamente inizializzata. Il task deve definire la funzione tramite una struttura Interrupt e memorizzare l'indirizzo di questa struttura nel parametro io_Data della struttura IOStdReq utilizzata per inviare il comando; definendo la funzione, il task deve anche indicare nel parametro ln_Pri della struttura Interrupt la priorità da assegnare alla funzione. Compiute queste operazioni, invia il comando KBD_ADDRESETHANDLER, il quale aggiunge la funzione alla lista delle funzioni di gestione del reset, nella posizione determinata dal parametro ln_Pri.

L'esito del comando viene indicato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato, mentre IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità 0; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando KBD_ADDRESETHANDLER e inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che ha successo o meno a seconda delle condizioni

in cui si trova il sistema al momento dell'invio del comando). Deve inoltre inizializzare io_Data in modo che punti alla struttura Interrupt che individua i codici e l'area dati della nuova funzione da aggiungere alla lista di reset. Questa struttura Interrupt dev'essere inizializzata in modo che il parametro is_Code contenga l'indirizzo del punto d'ingresso della funzione e il parametro is_Data contenga l'indirizzo dell'area dati utilizzata dalla funzione.

Discussione

KBD_ADDRESETHANDLER aggiunge una nuova funzione di reset alla lista mantenuta dal sistema. La posizione della nuova funzione nella lista viene determinata dal parametro ln_Pri che il task deve aver opportunamente inizializzato. Questo parametro appartiene alla sotto-struttura Node, che a sua volta è contenuta nella struttura Interrupt della nuova funzione.

La funzione dev'essere strutturata in modo che possa essere chiamata con la seguente istruzione:

HandlerFunction (handlerData)
A1

HandlerFunction è il nome della funzione di gestione del reset da tastiera e rappresenta l'indirizzo del suo punto d'ingresso; si tratta dell'indirizzo che il task deve indicare nel parametro is_Code della struttura Interrupt che definisce la nuova funzione. L'argomento handlerData è il puntatore all'area dati che la funzione deve impiegare, ovvero il valore indicato nel parametro is_Data della struttura Interrupt.

Una funzione di questo tipo viene mandata in esecuzione quando l'utente imposta la combinazione di reset Ctrl/Amiga-Sinistro/Amiga-Destro. La funzione rimane attiva, cioè eseguibile in caso di reset, fino a quando non viene impartito il comando KBD_REMRESETHANDLER per rimuoverla dalla lista che raccoglie le funzioni di gestione del reset da tastiera.

Tutti gli eventi di reset possono essere intercettati da una serie di funzioni di gestione appositamente progettate. Il reset provocato dall'utente avvia l'esecuzione di queste funzioni nell'ordine determinato dalla loro priorità. Con questo tipo di organizzazione, ogni funzione di gestione del reset può eseguire una procedura di sicurezza per salvaguardare le attività del task che l'ha inserita nella lista di reset. Infatti la priorità delle suddette funzioni è sempre maggiore di quella assegnata alla procedura di reset standard, e la loro esecuzione avviene quindi prima che la macchina venga effettivamente sottoposta alla procedura di reset. Per esempio, se l'utente impone il reset da tastiera mentre è in corso un'operazione di scrittura su disco, e il task che ha richiesto l'accesso al disco ha aggiunto al sistema una propria funzione di gestione del reset, questa può portare a termine l'accesso al disco e compiere le necessarie operazioni di sicurezza prima che il controllo venga ceduto alla procedura di reset standard.

La funzione di gestione del reset aggiunta da un task deve fare in modo che le altre funzioni a priorità più bassa vengano anch'esse eseguite prima che la macchina venga sottoposta alla procedura di reset standard. Perché questo avvenga, prima di restituire il controllo deve inviare il comando KBD_RESETHANDLERDONE impiegando la funzione SendIO.

KBD_READEVENT

Scopo del comando

Il comando KBD_READEVENT permette a un task di ottenere gli eventi di input accumulati nel buffer di tastiera del dispositivo Keyboard. KBD_READEVENT si aspetta che il task abbia memorizzato nel parametro io_Data della struttura di I/O l'indirizzo di un buffer appositamente allocato, e nel parametro io_Length la sua lunghezza. In questo buffer il comando memorizza le strutture InputEvent presenti nel buffer di tastiera. La sua grandezza dev'essere un multiplo esatto del numero di byte occupati dalla struttura InputEvent. Se nel buffer di tastiera non sono presenti eventi da tastiera in attesa di elaborazione, il comando KBD_READEVENT non restituisce il controllo fino a quando non viene premuto almeno un tasto. Se invece alcuni eventi di tastiera sono presenti, ma non bastano a riempire il buffer, KBD_READEVENT restituisce ugualmente il controllo.

KBD_READEVENT prevede il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non viene accordato. L'esito del comando è indicato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato, mentre IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto. IOERR_BADLENGTH indica che il parametro io_Length è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità 0; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando KBD_READEVENT e inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che ha successo o meno a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando). Deve inoltre inizializzare

io_Length con la dimensione in byte del buffer di lettura definito dal task (per determinare la grandezza del buffer, occorre moltiplicare il numero di strutture InputEvent che si devono ricevere per la grandezza in byte della struttura stessa). Deve infine inizializzare io_Data in modo che punti al buffer in RAM destinato a contenere le strutture InputEvent.

Discussione

KBD_READEVENT permette al task di leggere e rimuovere gli eventi di input provocati dalla pressione dei tasti sulla tastiera. Questi eventi vengono immagazzinati in un buffer di tastiera fino a quando un task non provvede a leggerli. In condizioni normali c'è già l'AmigaDOS che accede al buffer interno del dispositivo Keyboard per intercettare gli eventi da tastiera e quindi se un task invia il comando KBD_READEVENT con l'intenzione di sovrapporsi all'AmigaDOS deve impostare una priorità d'esecuzione sufficientemente elevata.

Ogni tasto premuto viene definito da una struttura InputEvent, nella quale sono presenti i seguenti parametri:

- ie_NextEvent. Questo parametro punta alla successiva struttura InputEvent quando l'evento è concatenato all'interno di una lista; vale ovviamente 0 se la struttura è l'ultima della lista.

- ie_Class. Questo parametro indica la classe dell'evento: per gli eventi da tastiera vale IECLASS_RAWKEY.

- ie_SubClass. Questo parametro non viene utilizzato per gli eventi di input classificati come IECLASS_RAWKEY; le routine del dispositivo Keyboard lo impostano a 0.

- ie_Code. Questo parametro viene restituito con il codice del tasto la cui pressione, o rilascio, causa l'evento. A ogni tasto della tastiera dell'Amiga è assegnato un valore che lo identifica univocamente. I valori attualmente previsti vanno da 0x00 a 0x67. Per indicare se l'evento è stato causato dalla pressione o dal rilascio del tasto, il sistema azzera o imposta il bit 7 del parametro.

- ie_Qualifier. Ogni bit di questo parametro descrive ulteriormente l'evento, indicando se al momento della pressione (o del rilascio) del tasto è attivo uno dei tasti speciali dell'Amiga. Le associazioni fra i bit e i tasti speciali sono le seguenti: bit 0, tasto Shift di sinistra; bit 1, tasto Shift di destra; bit 2, CapsLock; bit 3, tasto Control; bit 4, tasto Alt di sinistra; bit 5, tasto Alt di destra; bit 6, tasto Amiga di sinistra; bit 7, tasto Amiga di destra. L'ultimo bit che viene preso in considerazione, il bit 8, viene impostato se il tasto appartiene alla tastierina numerica.

- ie_X, ie_Y e ie_TimeStamp. Questi parametri non vengono utilizzati per gli eventi di input classificati come IECLASS_RAWKEY; le routine del dispositivo Keyboard li impostano a 0.

Di solito gli eventi di input della tastiera vengono passati automaticamente al dispositivo Input (si veda il capitolo 7) e generalmente i task non impiegano il comando KBD_READEVENT per non correre il rischio di creare conflitti di priorità.

Le routine interne del dispositivo Keyboard in genere riescono a immagazzinare diversi eventi nel buffer di tastiera prima che questo venga svuotato da una lettura. Questi eventi continuano ad accumularsi finché un task non invia il comando KBD_READEVENT oppure il dispositivo Input non procede automaticamente a elaborarli. Se per un periodo di tempo prolungato né il task e né il dispositivo Input provvedono a leggere gli eventi di input da tastiera immagazzinati, il buffer può trovarsi in condizione di overflow; se questo accade, ogni altro tasto premuto dall'utente viene perduto.

KBD_READMATRIX

Scopo del comando

Il comando KBD_READMATRIX viene utilizzato per leggere la matrice dei tasti e inserirla nel buffer definito dal task. Questa "matrice" definisce lo stato del tasto (premuto o rilasciato) e viene automaticamente aggiornata dalle routine del dispositivo Keyboard ogni volta che l'utente interagisce con la tastiera. Per descriverla, il task utilizza i parametri io_Length e io_Data della struttura IOStdReq (opportunamente inizializzati): io_Data deve individuare in memoria il buffer definito dal task per contenere la matrice dei tasti quando la lettura dei dati è finita, mentre io_Length deve definirne la dimensione in byte.

KBD_READMATRIX prevede il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non ha successo. L'esito del comando viene indicato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto, mentre IOERR_BADLENGTH indica che è stato specificato in modo non corretto il parametro io_Length.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit

devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità 0; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando KBD_READMATRIX e inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che ha successo o meno a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando). Deve inoltre inizializzare io_Data in modo che punti al buffer da riempire con i valori della matrice; deve infine inizializzare io_Length con la dimensione del buffer (se questo valore non è adeguato, il buffer entra in overflow e la RAM adiacente all'area allocata per il buffer viene sovrascritta).

Discussione

KBD_READMATRIX è l'unico comando che interagisce con la matrice dei tasti. Legge i valori ivi contenuti e li memorizza nel buffer definito dal task. Le routine interne del dispositivo Keyboard aggiornano automaticamente il contenuto della matrice ogni volta che l'utente preme un tasto.

La matrice dei tasti contiene una serie di byte disposti secondo una sequenza predefinita. Ogni byte consiste di otto bit che rappresentano lo stato del tasto (premuto o rilasciato). I parametri io_Length e io_Data della struttura IOStdReq definiscono il numero di byte nella matrice dei tasti e la locazione in RAM del buffer a essa destinato; questi parametri devono essere inizializzati prima che il comando KBD_READMATRIX venga inviato al dispositivo. Una volta che tramite il comando KBD_READMATRIX il task ha ricevuto nel proprio buffer la matrice dei tasti, può procedere ad analizzarne ogni singolo bit per conoscere lo stato del tasto premuto.

Si adotti la seguente procedura per organizzare il proprio task in modo che possa rilevare lo stato di un tasto:

1. Utilizzare lo schema dei tasti della tastiera riportato nell'*Amiga ROM Kernel Reference Manual* per determinare il codice grezzo corrispondente a ciascun tasto (si ricordi che in quelle tavole i codici grezzi sono espressi in notazione esadecimale); il codice grezzo è il numero del bit a cui si deve accedere. Per esempio, il manuale mostra che al tasto funzione F2 è assegnato il codice grezzo 0x51, e quindi si deduce che nella matrice dei tasti lo stato di questo tasto è rappresentato dal bit 0x51.

2. Identificare il byte della matrice che contiene il bit desiderato presupponendo che la numerazione dei byte e quella dei bit inizino entrambe da 0. Per esempio, nel caso del tasto funzione F2 questa operazione si effettua dividendo il codice grezzo del tasto per 8 e prendendo in considerazione il valore intero del quoziente (il valore intero della divisione 0x51/0x08 è 0x0A; in decimale, il valore intero della divisione 81/8 è 10). Nel nostro esempio, quest'operazione indica

che il bit corrispondente al codice grezzo 0x51 si trova nel byte 10 della matrice dei tasti.

3. Identificare il bit all'interno del byte. È sufficiente calcolare il modulo 8 del codice grezzo (cioè, il resto della divisione fra il codice grezzo e la costante 8). Nel nostro esempio, il resto della divisione è 1, che indica il bit 1 del byte 10 all'interno della matrice dei tasti. A questo punto il task può accedere al bit corrispondente al codice grezzo e rilevarne lo stato. Un valore del bit pari a 0 indica che il tasto è rilasciato; 1 indica che il tasto è premuto.

KBD_REMRESETHANDLER

Scopo del comando

Il comando KBD_REMRESETHANDLER rimuove una funzione di gestione del reset dalla lista di sistema. Le funzioni rimosse dal comando sono quelle precedentemente aggiunte a questa lista tramite il comando KBD_ADDRESETHANDLER, al fine di permettere ai task di gestire il reset da tastiera prima dell'AmigaDOS.

KBD_REMRESETHANDLER viene sempre trattato come una richiesta di I/O accodato e quindi viene sempre restituito alla reply port del task che lo ha inviato. L'esito del comando viene indicato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato, mentre IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità 0; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando KBD_REMRESETHANDLER e inizializzare io_Flags a 0. Deve inoltre inizializzare io_Data con l'indirizzo della struttura Interrupt che rappresenta la funzione che si desidera rimuovere dal sistema, nella quale il parametro is_Code individua il punto d'ingresso della funzione e il parametro is_Data punti all'area dati da essa utilizzata.

Discussione

I comandi KBD_ADDRESETHANDLER e KBD_REMRESETHANDLER sono fra loro complementari; il primo aggiunge al sistema nuove funzioni di gestione del reset, mentre il secondo le rimuove. Entrambi utilizzano la struttura Interrupt per individuare i codici e i dati che definiscono la funzione da aggiungere o rimuovere.

KBD_RESETHANDLERDONE

Scopo del comando

KBD_RESETHANDLERDONE viene impartito all'interno di una funzione di gestione del reset per indicare al sistema che la funzione ha terminato la propria esecuzione. KBD_RESETHANDLERDONE informa il sistema che può mandare in esecuzione la funzione successiva nella lista di reset.

Il parametro io_Data della struttura IOStdReq dev'essere inizializzato con l'indirizzo della struttura Interrupt che definisce la funzione. L'esito del comando viene indicato nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato, mentre IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

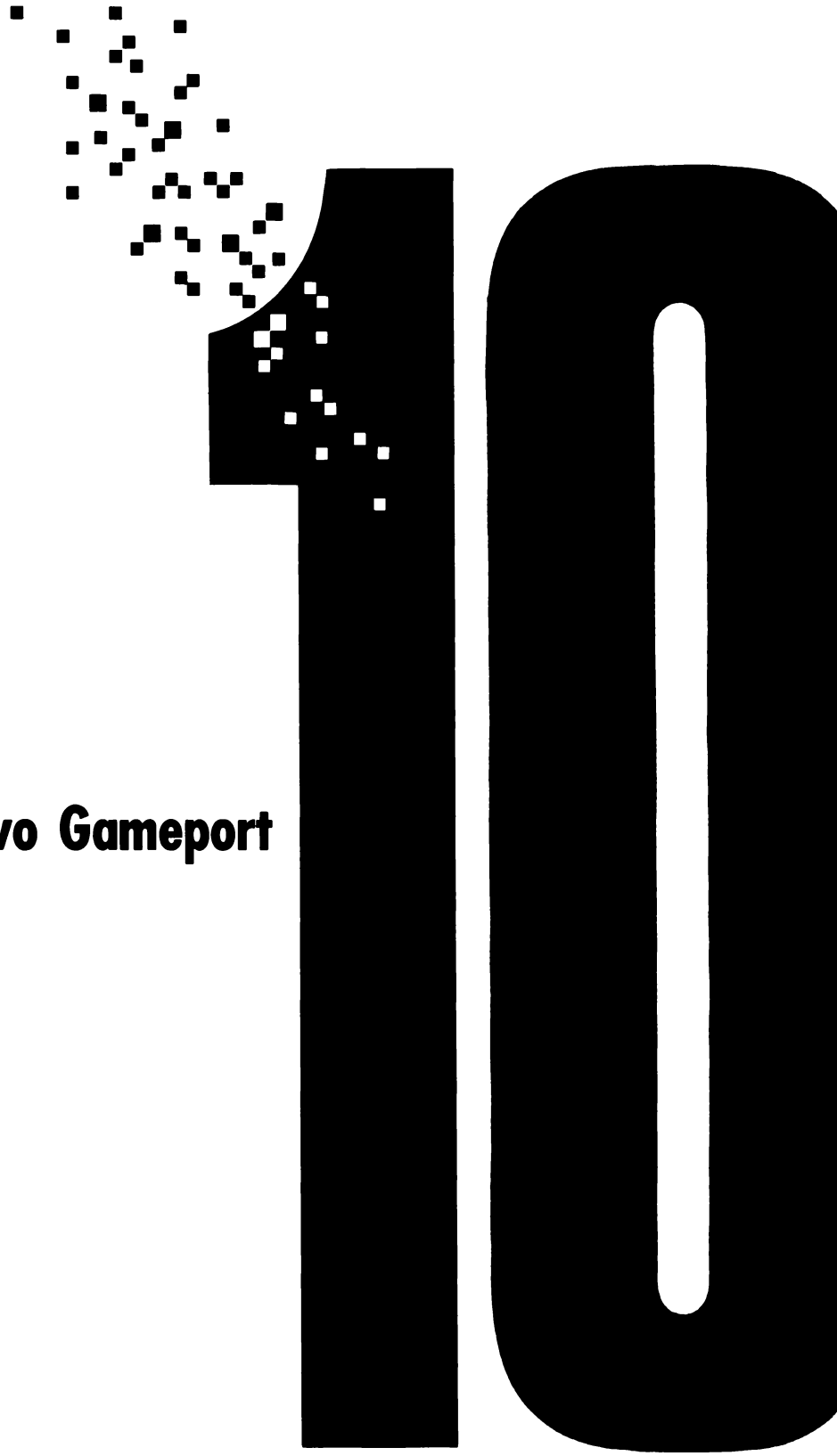
Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità 0; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando KBD_RESETHANDLERDONE e inizializzare io_Flags a 0. Deve inoltre inizializzare io_Data in modo che punti alla struttura Interrupt che rappresenta la funzione di gestione del reset.

Discussione

KBD_RESETHANDLERDONE serve alla funzione di gestione del reset per indicare al sistema che può mandare in esecuzione la funzione successiva, cioè quella con la priorità immediatamente inferiore. Se la funzione che esegue questo comando è l'ultima della lista, il controllo viene ceduto alla procedura di reset standard della macchina.

KBD_RESETHANDLERDONE deve sempre essere inviato tramite la funzione asincrona SendIO. La funzione di gestione del reset è una routine di interrupt software e non è possibile imporre uno stato d'attesa all'interno di una routine di interrupt software (come invece avverrebbe utilizzando la funzione sincrona DoIO).



Il dispositivo Gameport

Introduzione

Il dispositivo Gameport riunisce le informazioni provenienti da un dispositivo hardware collegato a una delle due porte giochi dell'Amiga. Il dispositivo Gameport risiede su ROM; nel caso dell'Amiga 1000 viene caricato automaticamente dal disco del Kickstart nella ROM WCS.

I due connettori delle porte giochi sono collocati nell'Amiga 1000 sul lato destro del cabinet, sul lato frontale in basso a destra nell'Amiga 2000 e sul retro nell'Amiga 500. Si tratta di prese DB-9 maschio da 9 pin. Il connettore di sinistra corrisponde all'unità 0 del dispositivo Gameport ed è normalmente adibito a ricevere le informazioni inviate dal mouse. Il pulsante di selezione del mouse (il pulsante 1) è collegato al pin 6 del connettore, mentre il pulsante che visualizza la barra menu (il pulsante 2) è collegato al pin 9. La Tavola 10.1 (nella pagina successiva) illustra il significato di ciascun pin presente nei due connettori. Nell'analisi della tavola occorre tener presente che le specifiche indicate valgono anche per un dispositivo hardware di tipo trackball, che dal punto di vista software viene controllato come se fosse un mouse. Il joystick di tipo proporzionale, invece, attualmente non è gestito dal software sistema.

Le routine interne del dispositivo Gameport ricevono segnali grezzi dai connettori delle porte giochi e li convertono in una serie di eventi di input. Come accade per gli altri eventi di input riconosciuti dal sistema dell'Amiga, ogni evento del dispositivo Gameport viene rappresentato da una struttura InputEvent (per una descrizione della struttura InputEvent si veda il capitolo 7).

Le routine interne del dispositivo gestiscono un buffer separato per ciascuna delle due unità (quindi per ciascuna delle due porte giochi). Il dispositivo impiega questi buffer interni per non perdere gli eventi di input che non vengono immediatamente prelevati dal sistema. Questi due buffer vengono gestiti automaticamente dalla routine interne del dispositivo Gameport e non devono essere confusi con i buffer definiti dal task e utilizzati dal comando GPD_READEVENT.

Si osservi che nell'interazione con il dispositivo l'unità 0 corrisponde alla porta giochi 1 e l'unità 1 corrisponde alla porta giochi 2.

Vediamo cosa accade se il task utilizza il comando GPD_READEVENT per trasferire le strutture InputEvent dal buffer dell'unità 0 nel suo buffer interno. Il task invia la richiesta di I/O che definisce il comando, la quale viene accodata alla request port dell'unità. Se nella coda alla request port non si trovano altre richieste, il comando inviato dal task viene soddisfatto non appena l'utente provoca un evento di input agendo sul mouse. Se invece, come accade normalmente per l'unità 0, nella coda alla request port esistono già richieste di comandi GPD_READEVENT inviate da qualche altro task nel sistema (in genere dal task di input del dispositivo Input), il nostro task non riceve risposta finché la nostra richiesta non ha completato l'ascesa lungo la coda. Quindi, a meno di non appropriarsi dell'intera macchina, un task non può riuscire a ottenere tutti gli eventi di input prodotti dell'unità 0 del dispositivo Gameport.

Pin	Joystick assoluto	Mouse	Joystick relativo	Dispositivi proporzionali
1	Avanti	Impulso verticale	Inutilizzato	Inutilizzato
2	Indietro	Impulso orizzontale	Inutilizzato	Inutilizzato
3	Sinistra	Impulso quadratura verticale	Pulsante 1	Pulsante sinistro
4	Destra	Impulso quadratura orizzontale	Pulsante 2	Pulsante destro
5	Inutilizzato	Pulsante 3 (eventuale)	Potenziometro X	Potenziometro di destra
6	Pulsante di sparo	Pulsante 1	Inutilizzato	Inutilizzato
7	+5 volt	+5 volt	+5 volt	+5 volt
8	Massa	Massa	Massa	Massa
9	Pulsante 2 (eventuale)	Pulsante 2	Potenziometro Y	Potenziometro di sinistra

Tavola 10.1: *Significato dei pin dei connettori delle porte giochi*

Si tenga presente, inoltre, che quelli che invece riesce a rimuovere non giungono a nessun altro task: nella fattispecie non vengono ricevuti dal dispositivo Input, e quindi neanche da Intuition. L'ovvia conseguenza è che il puntatore sullo schermo, controllato da Intuition, non si muove quando il nostro task intercetta un evento di input relativo a un cambiamento di stato del mouse. In condizioni normali, se il task accede invece all'unità 1 (alla quale il dispositivo Input in genere non accede) dovrebbe riuscire ad appropriarsi di tutti gli eventi di input generati dal dispositivo collegato a quella porta (sempre che non esistano altri task che vi accedono).

Se invece il task non trasferisce le strutture InputEvent nel proprio buffer, queste vengono tutte prelevate dal dispositivo Input se provengono dalla porta giochi 1, oppure continuano ad accumularsi se sono prodotte da un dispositivo collegato alla porta giochi 2 (il buffer non è comunque in grado di immagazzinare più di otto strutture InputEvent: i successivi eventi vengono ignorati). Per quanto riguarda l'unità 0 (porta giochi 1), il dispositivo Input vi accede costantemente e raggruppa in un unico flusso gli eventi di input che ottiene dalla porta insieme a quelli prodotti dai dispositivi Keyboard, TrackDisk e da qualsiasi altra sorgente nel sistema dell'Amiga (quindi anche dai task). Il flusso di eventi viene quindi ceduto alle funzioni di gestione degli input secondo l'ordine imposto dalle rispettive priorità (si ricordi che la funzione di

gestione degli eventi di input di Intuition possiede priorità 50, mentre quella del dispositivo Console possiede priorità 0). Si veda il capitolo dedicato al dispositivo Input per maggiori informazioni sul flusso degli eventi di input e sulle relative funzioni di gestione.

I comandi GPD_ASKCTYPE e GPD_SETCTYPE permettono a un task di avere informazioni e d'impostare il tipo di dispositivo hardware collegato a una delle porte giochi. Questi due comandi consentono alle routine interne d'interpretare correttamente i segnali provenienti dalla periferica. Inoltre, i comandi GPD_ASKTRIGGER e GPD_SETTRIGGER permettono a un task di avere informazioni e d'impostare quattro condizioni di trigger, cioè condizioni che causano la generazione di un evento di input. Le condizioni di trigger previste riguardano i cambiamenti di stato dei pulsanti del mouse, l'azzeramento di un conto alla rovescia, un movimento orizzontale del mouse superiore a una lunghezza data e infine un movimento verticale del mouse superiore a una lunghezza data. Quindi, se il task invia un comando GPD_READEVENT la sua struttura di I/O viene restituita solo quando si verifica una delle condizioni suddette (ovvero quando si verifica un evento di input).

Per inviare i comandi al dispositivo Gameport occorre una struttura di tipo IOStdReq, la struttura di I/O estesa standard prevista dalla maggior parte dei dispositivi. Alcuni comandi prevedono anche l'uso di altre strutture. GPD_READEVENT restituisce nel buffer indicato dal task una o più strutture InputEvent, la cui quantità dipende dalla capienza del buffer e dalla quantità di eventi che si sono accumulati nel buffer interno dell'unità. Il comando GPD_ASKTRIGGER restituisce nella struttura GamePortTrigger indicata dal task le condizioni di generazione degli eventi di input. Infine, il comando GPD_SETTRIGGER si aspetta che nel parametro io_Data della struttura di I/O sia presente l'indirizzo di una struttura di tipo GamePortTrigger nella quale il task abbia memorizzato le condizioni di generazione degli eventi di input. Si noti che la struttura GamePortTrigger è specifica del dispositivo Gameport.

Funzionamento del dispositivo Gameport

La Figura 10.1 (nella pagina successiva) illustra le operazioni che caratterizzano il dispositivo Gameport. I segnali di controllo provenienti dalla porta giochi di sinistra vengono sempre inviati all'unità 0 del dispositivo, mentre quelli provenienti dalla porta giochi di destra vengono inviati all'unità 1. Entrambe le unità possono essere aperte soltanto nel modo di accesso condiviso.

Il dispositivo Input, o meglio, il suo task di input, si aspetta che il mouse sia connesso alla porta giochi 1, e quindi invia comandi GPD_READEVENT all'unità 0 del dispositivo Gameport. Un task può però inviare il comando IND_SETMPORT al dispositivo Input per comunicare al sistema che il mouse è connesso alla porta giochi 2. In questo caso il task di input procede a inviare comandi GPD_READEVENT all'unità 1; si tratta comunque di una situazione atipica, o quanto meno non di default, e quindi nel seguito presupporremo sempre che il sistema consideri il mouse collegato alla porta giochi 1 (l'unità

0). Tutti gli eventi generati dall'unità 0 vengono quindi raggruppati dalle routine interne del dispositivo Input nel flusso globale degli eventi di input; il task può utilizzare il comando GPD_READEVENT per elaborare gli input provenienti dall'unità 0, ma deve tener conto che quest'azione interferisce con il sistema e non riuscirebbe in ogni caso a ottenere tutti gli eventi che l'utente può provocare agendo sul mouse.

Gli eventi di input del dispositivo Gameport generati dall'unità 1 non vengono generalmente presi in considerazione dal task di input del dispositivo Input. Se un task accede in lettura all'unità 1, con ogni probabilità non interferisce in nessuna delle attività condotte dal sistema, e quindi riesce a ottenere tutti gli eventi di input che da quella porta vengono generati.

Inviando il comando GPD_READEVENT a una delle due unità, tutte le strutture InputEvent presenti nel buffer dell'unità vengono trasferite nel buffer indicato dal task. Il task può quindi elaborare questi eventi direttamente, nel modo che il programmatore ritiene opportuno; gli eventi in questione non giungeranno a nessun altro task.

Il dispositivo Gameport è in grado d'interpretare segnali generati dai seguenti tipi di periferiche:

- un mouse, selezionabile indicando la costante GPCT_MOUSE quando si invia il comando GPD_SETCTYPE. Questo dispositivo hardware può generare eventi di input relativi a uno, due o tre pulsanti, e ai movimenti

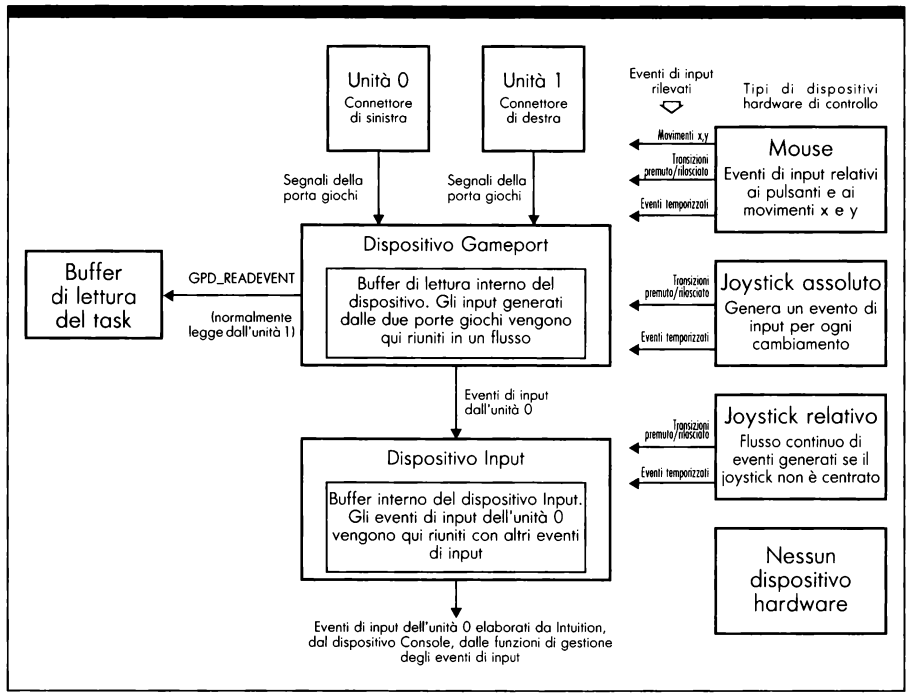


Figura 10.1: *Funzionamento del dispositivo Gameport*

del mouse. Questi eventi si suddividono in due categorie: 1) movimenti (in avanti o all'indietro) lungo le coordinate X e Y che generano offset relativi alla posizione precedente del mouse; 2) variazioni dello stato dei tasti (da premuti a rilasciati e viceversa).

- Un joystick assoluto, selezionabile indicando la costante GPCT_ABSJOYSTICK quando s'invia il comando GPD_SETCTYPE. Questo tipo di dispositivo hardware provoca un evento di input per ogni cambiamento di stato del joystick. Questi eventi si suddividono in due categorie: 1) variazioni dello stato del pulsante di sparo (da premuto a rilasciato e viceversa); 2) movimenti della cloche nelle otto direzioni.

- Un joystick relativo, selezionabile indicando la costante GPCT_RELJOYSTICK quando si invia il comando GPD_SETCTYPE. Questo dispositivo hardware genera un flusso continuo di eventi di input se lo stato del joystick non è quello di riposo (cloche in posizione centrale e pulsante di sparo rilasciato). Questi eventi di input si suddividono in due categorie: 1) variazioni dello stato del pulsante di sparo (da premuto a rilasciato e viceversa); 2) movimenti della cloche nelle otto direzioni.

Oltre a questi dispositivi hardware, il file INCLUDE gameport.h ne definisce uno denominato GPCT_NOCONTROLLER. Indicando questa costante quando s'invia il comando GPD_SETCTYPE, il dispositivo Gameport è informato che nessun tipo di dispositivo hardware è connesso alla porta indicata e che deve quindi ignorare gli eventuali segnali provenienti da quella porta.

Per ognuno dei tre dispositivi hardware citati, il dispositivo Gameport è anche in grado di generare eventi ogni volta che scade un conto alla rovescia prefissato dal task, che viene sempre avviato dopo la generazione dell'evento. Questo significa che i task possono richiedere al dispositivo Gameport di generare un evento quando l'utente non ne produce uno entro un certo tempo.

Elaborazione degli eventi di input prodotti dal dispositivo Gameport

La Figura 10.2 (nella pagina successiva) descrive l'origine degli eventi di input prodotti dal dispositivo Gameport e i loro possibili percorsi nel sistema. Gli eventi hanno origine nell'hardware della porta giochi sotto forma di segnali che le routine interne del dispositivo Gameport sondano per l'elaborazione. La sequenza generale di elaborazione risulta molto simile a quella che caratterizza il dispositivo Keyboard (si veda la Figura 9.2 a pagina 306), se si esclude il fatto che l'AmigaDOS non si appropria di alcun evento di input proveniente dalle porte giochi.

Gli eventi vengono generati se i segnali provenienti dalla porta soddisfano almeno una delle condizioni di trigger. Ogni evento è rappresentato da una struttura InputEvent che ne descrive le caratteristiche, in conformità con il tipo di periferica collegata. Ogni struttura InputEvent viene riposta nel buffer

interno dell'unità, in modo che il dispositivo possa rilevare altri eventi anche se nessun task elabora quelli già pervenuti. La capienza massima dei buffer delle unità è di otto strutture InputEvent, limite oltre il quale il dispositivo Gameport non genera altri eventi e attende che un task acceda in lettura all'unità.

I task inviano il comando GPD_READEVENT per leggere gli eventi presenti nei buffer delle unità. Quando il comando raggiunge la sommità della coda alla request port e ci sono diversi eventi in attesa di essere rimossi dal buffer interno, il task che ha inviato il comando li riceve tutti nel proprio buffer (sempre che lo spazio sia sufficiente). Gli eventi che il task ottiene diventano di sua proprietà e non vengono più intercettati da nessun altro task.

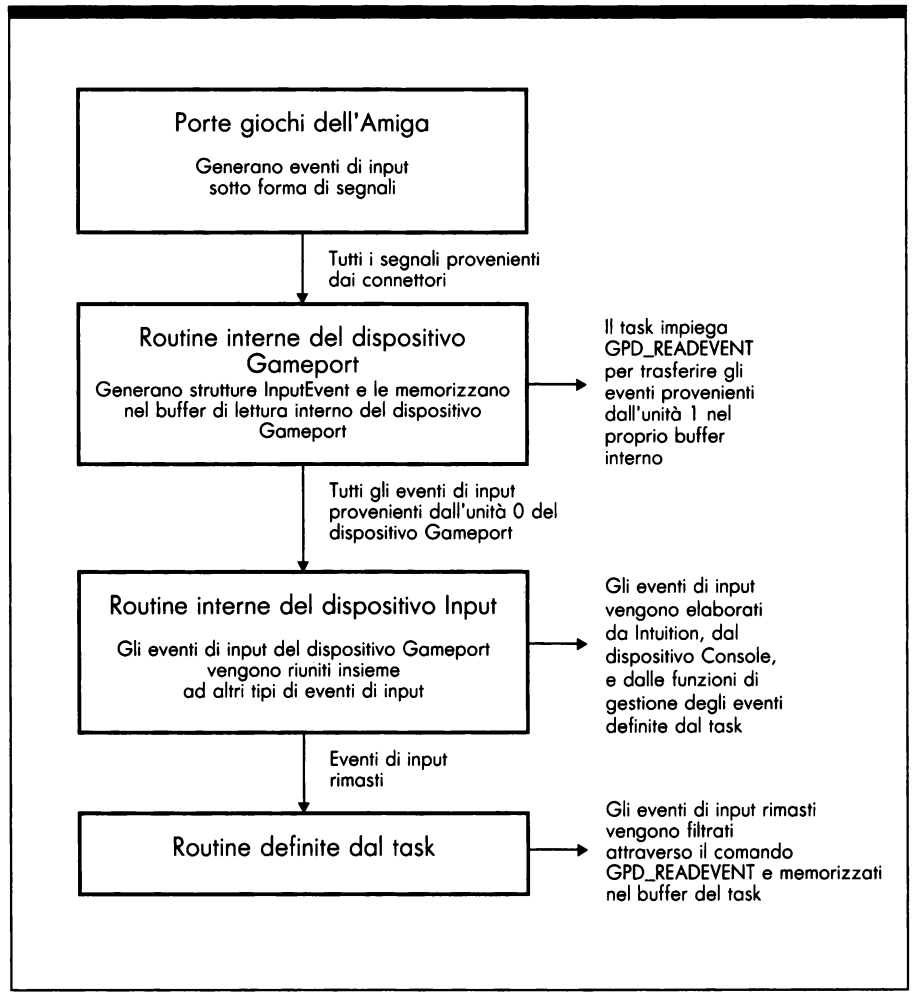


Figura 10.2: *Elaborazione degli eventi di input eseguita dal dispositivo Gameport*

comandi del dispositivo Gameport

Per programmare il dispositivo Gameport si possono utilizzare cinque comandi specifici e un comando standard; quattro sono immediati (GPD_ASKCTYPE, GPD_ASKTRIGGER, GPD_SETCTYPE, GPD_SETTRIGGER), mentre due consentono il QuickIO (CMD_CLEAR, GPD_READEVENT). Tutti i comandi influenzano il parametro io_Error della struttura IOStdReq; il comando CMD_CLEAR influenza anche i buffer interni del dispositivo Gameport.

L'invio dei comandi al dispositivo Gameport

La Figura 10.3 (nella pagina successiva) descrive lo schema generale utilizzato per inviare comandi alle routine del dispositivo Gameport. Le linee con le frecce rappresentano i parametri da inizializzare e quelli che vengono restituiti dalle routine interne del dispositivo. L'interazione con il dispositivo Gameport prevede tre fasi.

1. *Preparazione della struttura IOStdReq*. In questa fase il task possiede un completo controllo. Qui si inizializzano i parametri standard della struttura IOStdReq, in preparazione dell'invio di un comando. La scelta dei parametri dipende dal tipo di comando che s'intende inviare.

2. *Invio del comando e sua elaborazione*. L'unico compito da parte del programmatore, in questa fase, consiste nell'inviare il comando al dispositivo utilizzando le funzioni BeginIO, DoIO o SendIO. Successivamente il controllo passa alle routine interne del dispositivo che procedono a soddisfare la richiesta.

3. *Elaborazione dei parametri e loro restituzione*. In questa fase le routine interne del dispositivo Gameport possiedono un completo controllo. I risultati prodotti dall'elaborazione del comando vengono restituiti alla reply port del task che l'ha inviato.

Se il QuickIO non ha successo, la richiesta di I/O viene elaborata nell'istante in cui raggiunge la sommità della coda alla request port e, a elaborazione ultimata, passa nella coda alla reply port del task. Se invece il QuickIO viene accordato, la richiesta – dopo l'elaborazione – viene restituita direttamente al task che ha inoltrato il comando.

Come si può notare osservando la figura, diversi comandi del dispositivo Gameport restituiscono dei parametri in seguito alle loro elaborazioni; inoltre, il comando GPD_READEVENT riempie il buffer del task (individuato dal puntatore io_Data della struttura IOStdReq) con le strutture InputEvent che caratterizzano l'ultima serie di eventi di input provenienti dalla porta giochi indirizzata.

La Figura 10.3 descrive inoltre i parametri che ricoprono un ruolo

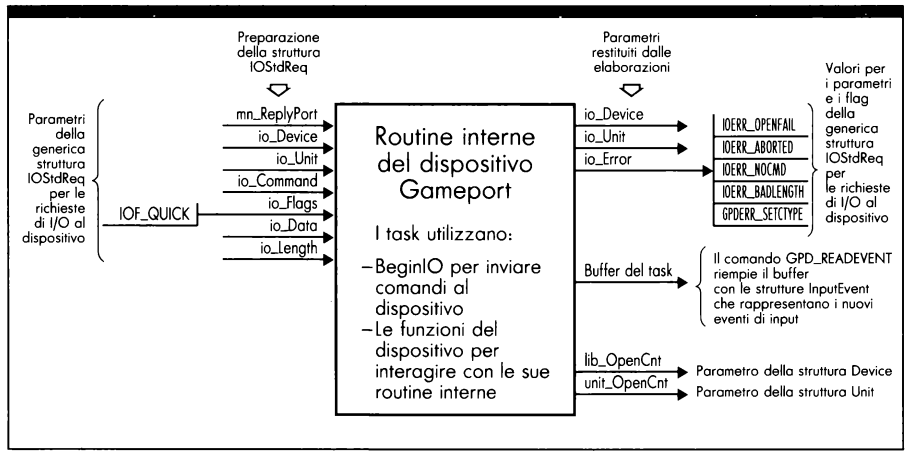


Figura 10.3: *Gestione delle funzioni e dei comandi previsti dal dispositivo Gameport*

significativo nella preparazione e nell'elaborazione delle funzioni del dispositivo Gameport. Le funzioni OpenDevice e CloseDevice influenzano il parametro lib_OpenCnt della struttura Device; OpenDevice influenza inoltre il parametro io_Error.

Le strutture del dispositivo Gameport

Il dispositivo Gameport utilizza una sola struttura speciale: GamePortTrigger. Viene impiegata dai comandi GPD_ASKTRIGGER e GPD_SETTRIGGER per rilevare e impostare le condizioni di trigger, cioè le condizioni che provocano la generazione di un evento di input.

La struttura GamePortTrigger

La struttura GamePortTrigger è definita come segue:

```
struct GamePortTrigger {
        UWORD gpt_Keys;
        UWORD gpt_Timeout;
        UWORD gpt_XDelta;
        UWORD gpt_YDelta;
};
```

I parametri della struttura GamePortTrigger hanno il seguente significato:

- gpt_Keys. Dev'essere inizializzato dal task a GPTF_DOWNKEYS se si desidera che venga generato un evento di input ogni volta che il

pulsante del mouse passa da rilasciato a premuto; a GPTF_UPKEYS se si desidera che venga generato un evento di input ogni volta che il pulsante del mouse passa da premuto a rilasciato. Queste due costanti sono rappresentate da due bit del parametro, che è possibile impostare simultaneamente qualora si desideri la generazione di un evento di input sia durante la pressione sia durante il rilascio del pulsante. Gpt_Keys agisce ovviamente nello stesso modo se la periferica connessa alla porta è un joystick.

- gpt_Timeout. Rappresenta l'intervallo di tempo che deve trascorrere prima che il dispositivo Gameport, in assenza di eventi generati dall'utente, ne generi uno automaticamente. Il conto alla rovescia viene avviato dopo la generazione dell'ultimo evento, e se prima dello scadere l'utente non produce un nuovo evento il dispositivo ne crea uno automaticamente e riavvia il conto alla rovescia. Questo intervallo è misurato in numero di "vertical blanking" (il vertical blanking individua il momento nel quale il pennello elettronico del tubo catodico, giunto alla fine di un quadro di schermo, si spegne per ritornare all'inizio della prima riga di schermo e generare un nuovo quadro; misurare il tempo in vertical blanking significa contare il numero di quadri che il pennello elettronico genera); in Europa sia il sistema PAL sia il sistema SECAM prevedono la generazione di 50 quadri al secondo, quindi l'unità di misura del parametro gpt_Timeout equivale a 1/50 di secondo. In tutti i Paesi nei quali viene invece impiegato il sistema NTSC, come gli Stati Uniti, le macchine generano 60 quadri al secondo, e quindi l'unità di misura del parametro gpt_Timeout equivale a 1/60 di secondo. A questo proposito, si noti che la frequenza di 50 o 60 Hz necessaria per pilotare il pennello elettronico nei due standard non dipende dalla frequenza di rete: l'hardware ricava la sua frequenza direttamente dal clock generato dal quarzo interno della macchina. Tale clock vale 7,1590 MHz per le macchine NTSC e 7,0939 MHz per le macchine PAL e SECAM. Se questo parametro vale 0, il dispositivo non produce alcun evento automaticamente.

- gpt_XDelta. Definisce la distanza orizzontale che dev'essere coperta dal mouse affinché venga generato un evento di input. Questa distanza è misurata in pixel di schermo.

- gpt_YDelta. Definisce la distanza verticale che dev'essere coperta dal mouse affinché venga generato un evento di input. Questa distanza è misurata in pixel di schermo.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOStdReq)
A1

Scopo della funzione

Questa funzione chiude l'accesso all'unità del dispositivo Gameport indicata nel parametro io_Unit della richiesta di I/O. Se con la chiamata alla funzione non rimangono unità aperte per il task e sono stati chiusi anche i dispositivi Console e Input, il dispositivo Gameport viene a sua volta chiuso per il task. In questo caso, CloseDevice inizializza i parametri io_Unit e io_Device della struttura IOStdReq a −1: il task non può più utilizzare questa struttura fino a quando questi parametri non vengono reinizializzati da una chiamata alla funzione OpenDevice, cioè fino a quando il task non riapre il dispositivo. CloseDevice provvede inoltre a decrementare il parametro lib_OpenCnt della struttura Device.

Argomenti della funzione

iOStdReq — Deve contenere l'indirizzo della struttura di tipo IOStdReq che il task impiega per interagire con il dispositivo. I suoi parametri, in particolare io_Device e io_Unit, vengono impostati dalla funzione OpenDevice al momento dell'apertura del dispositivo.

Discussione

CloseDevice chiude l'accesso di un task a una particolare unità del dispositivo. Un task dovrebbe sempre verificare che tutte le sue richieste di I/O inoltrate siano state restituite, prima di chiamare CloseDevice. Per farlo può utilizzare le funzioni GetMsg, Remove, CheckIO e WaitIO.

Quando un task ha concluso le proprie operazioni con il dispositivo Gameport, dovrebbe chiuderlo con una chiamata alla funzione CloseDevice in modo da liberare memoria per il sistema. Si ricordi che il dispositivo Gameport viene aperto indirettamente quando vengono aperti i dispositivi Input o Console, oppure quando viene aperto l'AmigaDOS. La procedura adottata per aprirlo determina il modo in cui si dovrebbe procedere per la sua chiusura.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("gameport.device", unitNumber, iOStdReq, ØL)
DØ                 AØ                  DØ          A1       D1
```

Scopo della funzione

La funzione OpenDevice apre per il task l'accesso a una delle due unità del dispositivo Gameport. Questo dispositivo viene aperto automaticamente dall'AmigaDOS o al momento dell'apertura dei dispositivi Console o Input.

OpenDevice inizializza automaticamente una struttura Unit per gestire l'unità prescelta del dispositivo Gameport. La struttura Unit contiene una sotto-struttura MsgPort rappresentante la request port dell'unità, e la relativa coda. Le unità del dispositivo Gameport operano nel modo di accesso condiviso, e quindi sono in grado di soddisfare le richieste di I/O provenienti da più task nel sistema.

OpenDevice oltre ad aprire l'unità indicata dal task memorizza nel parametro io_Device della struttura di I/O l'indirizzo della struttura Device di gestione del dispositivo e nel parametro io_Unit l'indirizzo della struttura Unit di gestione dell'unità. Incrementa inoltre il parametro lib_OpenCnt.

OpenDevice richiede una reply port inizializzata in modo appropriato, ed eventualmente un bit di segnale allocato dal task per essere avvisato nel momento in cui giunge una risposta inviata dal dispositivo Gameport. I risultati prodotti dall'esecuzione del comando vengono restituiti nei seguenti parametri:

- io_Device. In questo parametro OpenDevice memorizza l'indirizzo della struttura Device che il sistema impiega per interagire con la libreria di routine del dispositivo.

- io_Unit. In questo parametro OpenDevice memorizza l'indirizzo della struttura Unit che individua univocamente l'unità prescelta dal task.

- io_Error. Un valore di questo parametro pari a 0 indica che la richiesta di apertura del dispositivo ha avuto successo. IOERR_OPENFAIL indica che non è stato possibile aprire il dispositivo Gameport; questo tipo di errore normalmente si verifica se non c'è memoria sufficiente per aprire il dispositivo.

Argomenti della funzione

"gameport.device" Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Gameport.

unitNumber Il task deve indicare in questo argomento quale unità desidera aprire. Il dispositivo Gameport prevede l'unità 0 e l'unità 1. La funzione si aspetta che l'argomento sia una long word.

iOStdReq Dev'essere l'indirizzo della struttura di tipo IOStdReq che il task ha allocato e intende impiegare per interagire con il dispositivo.

ØL Indica che la funzione ignora l'argomento flag.

Preparazione della struttura IOStdReq

Per aprire il dispositivo Gameport occorre inizializzare il parametro mn_ReplyPort della struttura di I/O con l'indirizzo della struttura MsgPort che rappresenta la reply port del task. Il task può allocare una message port tramite la funzione CreatePort di supporto alla libreria Exec, e indicarne l'indirizzo come argomento della funzione CreateExtIO, sempre di supporto alla libreria Exec. Chiamando quest'ultima funzione il task alloca la struttura di I/O necessaria per interagire con il dispositivo e memorizza automaticamente l'indirizzo della reply port nel parametro mn_ReplyPort.

Discussione

La funzione OpenDevice apre per un task l'accesso alle routine interne del dispositivo Gameport. Questo dispositivo può essere aperto soltanto nel modo di accesso condiviso. Si ricordi inoltre che il task di input del dispositivo Input accede per default all'unità 0; la situazione può comunque essere capovolta

indicando al sistema che il dispositivo hardware di controllo del puntatore è connesso alla porta giochi 2 (unità 1).

COMANDI STANDARD DEL DISPOSITIVO

CMD_CLEAR

Scopo del comando

CMD_CLEAR azzera uno dei due buffer di lettura del dispositivo Gameport. Questi buffer vengono utilizzati per immagazzinare gli eventi di input del dispositivo Gameport (sotto forma di strutture InputEvent) in attesa che un task li prelevi tramite il comando GPD_READEVENT. Un task può inviare il comando CMD_CLEAR per assicurarsi che il buffer non contenga eventi di input indesiderati

CMD_CLEAR permette il QuickIO, e viene restituito alla reply port soltanto se il modo di accesso veloce non ha avuto successo. Nel parametro io_Error viene restituito il codice d'errore IOERR_NOCMD se il parametro io_Command è stato inizializzato in modo non corretto.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando CMD_CLEAR, e inizializzare io_Flags a 0 oppure a IOF_QUICK per il QuickIO (che ha successo o meno a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando). Si ricordi che la funzione DoIO richiede il QuickIO automaticamente.

Discussione

Il dispositivo Gameport mantiene automaticamente un buffer interno di

lettura per ogni unità, utilizzato per memorizzare gli eventi di input generati dal dispositivo hardware in comunicazione con l'unità prima che un task ne faccia richiesta. Il comando CMD_CLEAR azzera il buffer cancellando completamente il suo contenuto e serve quando un task vuole trasferire nel suo buffer gli eventi di input avvenuti da un certo istante in poi. Il dispositivo Gameport non possiede un comando CMD_WRITE e quindi non utilizza alcun buffer di scrittura.

COMANDI SPECIFICI DEL DISPOSITIVO

GPD_ASKCTYPE

Scopo del comando

Il comando GPD_ASKCTYPE informa il task su quale dispositivo hardware è collegato alla porta indicata. Nella locazione di memoria indicata dal parametro io_Data della struttura IOStdReq, le routine interne del dispositivo Gameport restituiscono una costante che identifica il tipo di dispositivo hardware collegato (si ricordi che il dispositivo Gameport non è in grado di stabilire che tipo di dispositivo hardware è collegato, quindi la costante è quella di default o quella impostata da un precedente comando GPD_SETCTYPE).

GPD_ASKCTYPE viene sempre eseguito in modo immediato e viene restituito alla reply port solo se non è stato richiesto il QuickIO. I risultati prodotti dall'esecuzione del comando vengono restituiti nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato. IOERR_NOCMD indica che il parametro io_Command non è stato specificato in modo corretto, e IOERR_BADLENGTH indica che è stato specificato in modo non corretto il parametro io_Length.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando GPD_ASKCTYPE; inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere

il QuickIO. Si devono inoltre inizializzare i seguenti parametri:

- io_Length. Si deve inizializzarlo a 1, per indicare che il valore restituito nella locazione indirizzata dal parametro io_Data è composto da un solo byte.

- io_Data. Si deve inizializzarlo con l'indirizzo della locazione di memoria (un byte) nella quale si memorizza l'informazione sul tipo di dispositivo collegato, ovvero uno dei cinque possibili valori che discuteremo nel prossimo paragrafo.

Discussione

I comandi GPD_ASKCTYPE e GPD_SETCTYPE permettono a un task rispettivamente di rilevare e d'indicare il tipo di dispositivo hardware collegato a un'unità del dispositivo Gameport. GPD_ASKCTYPE dice al task quale periferica è collegata alla porta. Il dispositivo, però, non è in grado di effettuare il controllo personalmente e quindi il valore restituito è quello di default o quello impostato tramite un comando GPD_SETCTYPE. Un task e il dispositivo Gameport devono sempre sapere quale tipo di dispositivo hardware è collegato alla porta, per poter interpretare correttamente i segnali da esso prodotti.

Il dispositivo Gameport è in grado di riconoscere le seguenti costanti: GPCT_NOCONTROLLER (nessun dispositivo hardware), GPCT_MOUSE (mouse o trackball), GPCT_RELJOYSTICK (joystick relativo) e GPCT_ABSJOYSTICK (joystick assoluto). Queste costanti sono definite nel file INCLUDE denominato gameport.h.

GPD_ASKTRIGGER

Scopo del comando

Il comando GPD_ASKTRIGGER permette a un task di sapere quali condizioni di trigger si devono verificare perché il dispositivo Gameport riconosca come eventi di input i segnali provenienti dall'hardware collegato. Il comando restituisce la risposta nella struttura di tipo GamePortTrigger indicata dal task nella richiesta di I/O.

GPD_ASKTRIGGER è un comando immediato, e viene restituito alla reply port del task soltanto se non è stato richiesto il QuickIO. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato abortito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in

modo non corretto e IOERR_BADLENGTH indica che è stato specificato in modo non corretto il parametro io_Length.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando GPD_ASKTRIGGER; inizializzare io_Flags a 0 oppure a IOF_QUICK per il QuickIO (che ha successo o meno a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando). Si devono inoltre inizializzare i seguenti parametri:

- io_Length. Deve contenere la lunghezza (in byte) della struttura GamePortTrigger. Un task può utilizzare l'operatore sizeof del linguaggio C per inizializzare questo parametro.

- io_Data. Deve contenere l'indirizzo della struttura GamePortTrigger opportunamente allocata dal task. Quando il comando ritorna, il task può accedere ai parametri di questa struttura per sapere quali sono le condizioni di trigger.

Discussione

I comandi GPD_ASKTRIGGER e GPD_SETTRIGGER permettono a un task di rilevare e di cambiare le condizioni di trigger per una porta giochi, cioè le condizioni che provocano la generazione di un evento di input.

GPD_READEVENT

Scopo del comando

Il comando GPD_READEVENT permette a un task di ricevere all'interno di un proprio buffer le strutture InputEvent che si trovano nel buffer interno dell'unità indicata. Ogni struttura InputEvent presente nel buffer descrive un evento di input che non è ancora stato letto (e rimosso) da nessun task.

GPD_READEVENT utilizza i parametri io_Length e io_Data della struttura IOStdReq per individuare il buffer definito dal task; la grandezza di questo buffer determina il numero di strutture InputEvent che il comando può ricevere. Generalmente si definisce il buffer in modo che possa contenere otto strutture InputEvent, il massimo che può contenere il buffer dell'unità. Un task può comunque indicare un buffer di dimensioni inferiori.

Se nel buffer dell'unità non è presente alcun evento di input, il comando GPD_READEVENT non viene restituito al task fino a quando non se ne verifica almeno uno. Il comando, comunque, non aspetta che il buffer si riempia completamente, ma preleva e trasferisce le strutture InputEvent presenti nel momento in cui entra in esecuzione. Il task può sapere quante strutture InputEvent ha ottenuto nel proprio buffer leggendo il contenuto del parametro io_Actual restituito dal comando: in questo parametro il dispositivo memorizza il numero di byte trasferiti.

GPD_READEVENT permette il QuickIO e viene restituito alla coda della reply port soltanto se il QuickIO non è stato accordato. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato. IOERR_NOCMD indica che il parametro io_Command non è stato specificato in modo corretto, e IOERR_BADLENGTH indica che non è stato specificato in modo corretto il parametro io_Length.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando GPD_READEVENT; inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO (che ha successo o meno a seconda delle condizioni in cui si trova il sistema al momento dell'invio del comando). Si devono inoltre inizializzare i seguenti parametri:

- io_Length. Deve contenere la dimensione (in byte) del buffer di lettura del task. Per determinare questo valore occorre moltiplicare il massimo numero di strutture che si vogliono ricevere per il numero di byte occupati dalla singola struttura. Un task può utilizzare l'operatore sizeof del linguaggio C per inizializzare questo parametro.

- io_Data. Deve contenere l'indirizzo del buffer definito dal task, nel quale GPD_READEVENT trasferisce le strutture InputEvent immagazzinate nel buffer interno dell'unità.

Discussione

GPD_READEVENT trasferisce nel buffer del task (sempre che le sue dimensioni siano sufficienti) tutte le strutture InputEvent che si trovano nel buffer dell'unità. Le strutture ricevute dal task non possono giungere a nessun altro task nel sistema. Ogni struttura InputEvent che arriva al buffer del task contiene i seguenti parametri:

- ie_NextEvent. Contiene l'indirizzo della successiva struttura InputEvent nella lista mantenuta dall'unità. Quando il task le riceve nel proprio buffer, le strutture InputEvent sono ordinate sequenzialmente e quindi questo parametro non ha più valore come indirizzo. Il task può comunque leggerlo per sapere se una struttura contiene il valore 0 (cioè se è l'ultima della lista-buffer); in altre parole, per sapere se il comando è riuscito a trasferire nel buffer del task tutte le strutture InputEvent che si trovavano nel buffer dell'unità.

- ie_Class. Indica la classe dell'evento di input proveniente dalla porta giochi; viene inizializzato a IECLASS_RAWMOUSE nel caso di movimenti generici effettuati da un mouse o da un joystick.

- ie_SubClass. Viene impostato a 0 se l'evento di input del dispositivo Gameport proviene dalla porta giochi di sinistra (unità 0); viene impostato a 1 se l'evento proviene dalla porta giochi di destra (unità 1).

- ie_Code. Questo parametro contiene il valore 0xFF (−1) se l'evento non è stato prodotto dal pulsante del mouse o del joystick; il valore 0x68 se l'evento è stato prodotto dalla pressione del pulsante sinistro; il valore 0x69 se l'evento è stato prodotto dalla pressione del pulsante destro; il valore 0x6A se l'evento è stato prodotto dalla pressione del pulsante centrale (per i dispositivi hardware che ne dispongono). Il rilascio del pulsante viene indicato dagli stessi valori, ma con il bit 7 impostato (rispettivamente, 0xE8, 0xE9, 0xEA). Le costanti IECODE_LBUTTON, IECODE_RBUTTON, IECODE_MBUTTON, corrispondono ai valori per i pulsanti premuti, mentre la costante IECODE_NOBUTTON indica che l'evento non è stato causato dalla pressione di un pulsante. Per maggiori informazioni si consulti il file INCLUDE inputevent.h.

- ie_Qualifier. Questo parametro è il qualificatore dell'evento di input proveniente dalla porta giochi. Può assumere i seguenti valori: IEQUALIFIER_LBUTTON, IEQUALIFIER_RBUTTON, IEQUALIFIER_MBUTTON oppure IEQUALIFIER_RELATIVEMOUSE (quest'ultimo bit viene impostato solo nel caso di un mouse o di un joystick relativo). Per maggiori informazioni si consulti il file INCLUDE inputevent.h.

- ie_X. Nel caso del mouse o del joystick relativo, questo parametro indica di quanti pixel si dovrebbe muovere orizzontalmente il puntatore sullo schermo (rispetto alla posizione attuale) per seguire i movimenti impressi dall'utente (gli offset negativi indicano direzioni verso sinistra). Nel caso del joystick assoluto, invece, questo parametro indica solo che il movimento avviene in direzione orizzontale: −1 indica verso sinistra, 1 indica verso destra. Infine, se l'evento non è stato prodotto da un movimento orizzontale, questo parametro viene azzerato.

- ie_Y. Nel caso del mouse o del joystick relativo, questo parametro indica di quanti pixel si dovrebbe muovere verticalmente il puntatore sullo schermo (rispetto alla posizione attuale) per seguire i movimenti impressi dall'utente (gli offset negativi indicano direzioni verso l'alto). Nel caso del joystick assoluto, invece, questo parametro indica solo che il movimento avviene in direzione verticale: −1 indica verso l'alto, 1 indica verso il basso. Infine, se l'evento non è stato prodotto da un movimento verticale, questo parametro viene azzerato.

- ie_TimeStamp. Indica il tempo che è trascorso dall'ultimo evento di input. L'unità di misura è il numero di scansioni del quadro di schermo che il pennello elettronico è riuscito a completare dall'ultimo evento di input (nel caso dell'Europa – sistema PAL o SECAM – è misurato quindi in cinquantesimi di secondo, nel caso dell'America – sistema NTSC – è misurato invece in sessantesimi di secondo). Le routine interne del dispositivo Gameport memorizzano il numero di quadri nel parametro tv_secs della struttura timeval. Si ricordi che in genere questo parametro indica un periodo di tempo espresso in secondi, che può essere completato dal valore contenuto nel parametro tv_micro (microsecondi). Tuttavia, in questo caso, il valore tv_micro non ha alcun significato: il dispositivo Gameport inizializza solo il parametro tv_secs con il numero di schermi visualizzati dal pennello elettronico dopo l'ultimo evento di input rilevato.

Gli eventi del dispositivo Gameport relativi alla porta giochi 1 (l'unità 0) vengono generalmente richiesti, tramite l'invio di comandi GPD_READEVENT, dal task di input del dispositivo Input, che in condizioni normali è il solo a interagire con la porta giochi 1 a un così basso livello. Tuttavia un qualunque task può inviare a sua volta comandi GPD_READEVENT alla stessa porta, entrando in competizione con il task di input nella lettura degli eventi. In questo caso, gli eventi che arrivano all'uno non arriveranno all'altro. Quando gli eventi di input sono intercettati dal task del programmatore, Intuition non può riceverli e quindi il puntatore sullo schermo non si muove.

Gli eventi di input vengono accodati nel buffer interno dell'unità fino a quando un task nel sistema non provvede a leggerli con il comando GPD_READEVENT. Se nessun task accede in lettura all'unità e il buffer raggiunge la sua capienza massima (otto strutture InputEvent), il dispositivo non genera più nuovi eventi finché un task non invia un comando GPD_READEVENT.

GPD_SETCTYPE

Scopo del comando

Il comando GPD_SETCTYPE permette a un task d'indicare al dispositivo Gameport che tipo di dispositivo hardware è collegato a una delle due porte giochi. Inviando questo comando, il task si assicura che il dispositivo interpreti correttamente i segnali in arrivo. Gameport attribuisce una costante per ciascuna delle periferiche che è in grado di riconoscere. Prima d'inviare il comando, il task deve memorizzare nel parametro io_Data della richiesta di I/O l'indirizzo di una locazione di memoria nella quale ha inserito la costante relativa alla periferica collegata. Queste costanti sono state analizzate nel paragrafo dedicato al comando GPD_ASKCTYPE.

GPD_SETCTYPE viene sempre eseguito in modo immediato, e viene restituito alla reply port soltanto se non è stato richiesto il QuickIO. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato. IOERR_NOCMD indica che il parametro io_Command non è stato specificato nel modo corretto, e IOERR_BADLENGTH indica che non è stato specificato in modo corretto il parametro io_Length. GPDERR_SETCTYPE indica che il dispositivo hardware indicato non risulta fra quelli riconosciuti dal dispositivo Gameport.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando GPD_SETCTYPE; inizializzare io_Flags a 0 oppure a IOF_QUICK se si desidera che la risposta non venga restituita alla reply port. Si devono inoltre inizializzare i seguenti parametri:

- io_Length. Deve contenere il valore 1 per indicare che la costante che identifica il tipo di dispositivo è lunga 1 byte.

- io_Data. Deve contenere l'indirizzo della locazione di memoria nella quale si è memorizzata la costante che rappresenta il tipo di dispositivo hardware collegato. Si veda la discussione del comando GPD_ASKCTYPE per maggiori informazioni sul significato di questi

valori, e si consulti il file INCLUDE gameport.h per le definizioni delle relative costanti.

Discussione

Il tipo di dispositivo hardware collegato all'unità dev'essere noto sia al dispositivo Gameport sia al task, in modo che entrambi siano in grado di decifrarne i segnali in modo corretto. GPD_SETCTYPE permette a un task d'informare il dispositivo Gameport circa il tipo di dispositivo hardware che deve considerare collegato a una delle due porte.

GPD_SETTRIGGER

Scopo del comando

Il comando GPD_SETTRIGGER permette a un task di cambiare le condizioni di trigger, cioè le condizioni che provocano la generazione di un evento di input. GPD_SETTRIGGER è un comando immediato, e viene restituito alla reply port del task soltanto se non è stato richiesto il QuickIO. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_ABORTED indica che il comando è stato eliminato. IOERR_NOCMD indica che il parametro io_Command non è stato specificato in modo corretto, e IOERR_BADLENGTH indica che non è stato specificato in modo corretto il parametro io_Length.

Preparazione della struttura IOStdReq

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono rispettivamente contenere l'indirizzo della struttura Device di gestione del dispositivo e l'indirizzo della struttura Unit di gestione dell'unità; questi indirizzi vengono restituiti dalla funzione OpenDevice al momento dell'apertura dell'unità. Il task deve impostare io_Command con il comando GPD_SETTRIGGER; inizializzare io_Flags a 0 oppure a IOF_QUICK per richiedere il QuickIO, se si desidera che la risposta non venga restituita alla reply port. Si devono inizializzare inoltre i seguenti parametri:

- io_Length. Deve contenere la dimensione (in byte) della struttura GamePortTrigger.

- io_Data. Deve contenere l'indirizzo della struttura GamePortTrigger nella quale sono state memorizzate le nuove condizioni di trigger per l'unità.

Discussione

I comandi GPD_ASKTRIGGER e GPD_SETTRIGGER permettono a un task di rilevare e di cambiare le condizioni di trigger per una porta giochi, cioè le condizioni che provocano la generazione di un evento di input.

Ogni volta che si verifica una condizione di trigger (o più d'una) le routine interne del dispositivo Gameport creano una nuova struttura InputEvent che caratterizza il nuovo evento e che viene immagazzinata nel buffer interno dell'unità.

Il dispositivo Printer

Introduzione

Il dispositivo Printer è responsabile dell'invio di caratteri stampabili e codici di controllo a una stampante collegata alla porta seriale o alla porta parallela. Per interfacciarsi correttamente con la stampante, il dispositivo Printer impiega i driver di stampa. Si tratta di particolari file all'interno dei quali il dispositivo Printer trova le informazioni di cui ha bisogno per comandare la stampante. Ogni driver di stampa nasce per comandare un particolare modello di stampante, privilegiando quelle che costituiscono standard diffusi e quelle che sfruttano al meglio le capacità grafiche dell'Amiga. Con la versione 1.3 del software sistema sono disponibili driver di stampa per le seguenti stampanti:

Alphacom AlphaPro 101	Epson Q	NEC Pinwriter
Brother HR-15XL	Epson XOld	Okidata 293I
CalComp ColorMaster	EpsonX[CBM MPS1250]	Okidata 92
CalComp ColorMaster2	Howtek Pixelmaster	Okimate 20
Canon PJ-1080A	HP DeskJet	Quadram QuadJet
CBM MPS1000	HP LaserJet	Qume LetterPro 20
Diablo 630	HP PaintJet	Toshiba P351C
Diablo Advantage D25	HP ThinkJet	Toshiba P351SX
Diablo C-150	Imagewriter II	Xerox 4020

Per le stampanti che non sono compatibili con nessuno dei driver indicati, esiste un apposito driver di stampa generico; con la versione 1.3 del Workbench questo driver si trova sul disco sistema, mentre gli altri risiedono tutti nel disco Extras 1.3. Si tenga presente che il driver di stampa per la stampante laser HP LaserJet può essere impiegato anche con le stampanti HP LaserJet Plus e HP LaserJet II.

Individuato il driver adatto alla propria stampante, l'utente lo seleziona tramite il tool Preferences; in seguito, ogni volta che un task, o l'utente stesso, chiama il dispositivo Printer, verrà impiegato quel particolare driver. Affinché i driver di stampa siano riconosciuti dal sistema, occorre che si trovino nella directory logica DEVS: (generalmente il sistema assegna al dispositivo logico DEVS: la directory devs del disco sistema; l'utente può comunque modificare questa scelta tramite il comando ASSIGN). Preferences accede a questa directory logica per elencare all'utente i driver di stampa disponibili, e il dispositivo Printer a sua volta si aspetta che nella subdirectory printers della directory logica DEVS: sia presente il driver prescelto (si ricordi che tutti i parametri di default alterabili tramite il tool Preferences sono contenuti nel file system-configuration, anch'esso nella directory logica DEVS:; quando l'utente attiva l'opzione SAVE del tool Preferences, tutti i parametri del sistema vengono salvati in questo file).

Per comunicare con una stampante si possono seguire fondamentalmente due strade. Si può trattarla come qualunque altra periferica, e quindi

comunicare con essa attraverso il dispositivo che gestisce la porta alla quale è collegata (Parallel o Serial). In questo caso il dispositivo Printer non viene aperto e i driver di stampa non vengono presi in considerazione. L'invio di dati alla stampante in questo modo non passa attraverso nessun filtro, come se i dati venissero inviati a un generico terminale esterno. Questa via può tornare utile quando si desidera che il sistema non intervenga minimamente sul flusso di dati diretti alla stampante, ma generalmente è sconsigliabile.

Il secondo metodo prevede l'impiego del dispositivo Printer, il quale per funzionare si avvale delle informazioni impostate dall'utente tramite il tool Preferences (relative al driver di stampa, alla porta a cui è collegata la stampante, al formato della pagina di stampa, al tipo di carta, alla qualità di stampa, ai margini, alla spaziatura e ad altri parametri che intervengono nella stampa in grafica); tutte queste informazioni sono elencate all'interno della struttura Preferences mantenuta dal sistema e modificata dal tool Preferences ogni volta che l'utente seleziona l'opzione Use o Save. Adottando questo secondo metodo, l'utente e i task riescono a gestire le operazioni di stampa con maggiore semplicità e con la sicurezza che i dati inviati siano nel formato previsto dalla stampante.

I dispositivi Serial, Parallel e Printer possono essere aperti e impiegati dai task con le normali procedure, ma possono essere anche impiegati da CLI (Command Line Interface, interfaccia linea comando), cioè direttamente dall'utente senza l'ausilio di alcun task. Per farlo, l'utente deve ridirigere verso l'appropriato dispositivo logico l'output che normalmente otterrebbe sullo schermo, oppure inviare un file al dispositivo tramite il comando COPY. I nomi dei dispositivi logici sono SER: per il dispositivo Serial, PAR: per il dispositivo Parallel, e PRT: per il dispositivo Printer. Per esempio, se l'utente desidera stampare velocemente il contenuto di un file di testo, con una formattazione funzionale ma ridotta al minimo, può inviare il comando:

COPY NomeFile TO PRT:

Questa operazione causa il caricamento del dispositivo Printer, del driver di stampa indicato nella struttura Preferences, e del dispositivo Serial o Parallel a seconda di quanto specificato dall'utente tramite il tool Preferences (si noti che quando le strutture Device di gestione dei dispositivi non compaiono nella lista di sistema DeviceList, i dispositivi vengono caricati automaticamente da disco). Se invece occorre stampare il contenuto di un file in valori esadecimali e non si desidera che intervenga alcun filtro, si può inviare il comando:

TYPE > PAR: NomeFile OPT h

Il dispositivo Printer risiede su disco, e dev'essere quindi caricato nella RAM per diventare operativo, cosa che avviene in modo trasparente quando OpenDevice rileva che nella lista di sistema DeviceList non risulta la relativa struttura Device di gestione. È da notare che l'apertura del dispositivo non ha successo qualora nella subdirectory printers della directory logica DEVS: non risulti presente il driver di stampa il cui nome è indicato nell'array PrinterFilename della struttura Preferences. Si tratta del nome impostato con

l'ultima esecuzione del tool Preferences, oppure ottenuto dal file di default DEVS:system-configuration durante l'attivazione della macchina. Nel caso che il dispositivo Printer non trovi su disco il driver di stampa indicato nella struttura Preferences, restituisce il codice d'errore IOERR_OPENFAIL e si autochiude. Non viene disallocato e quindi rimane in memoria, ma ogni volta che un task cerca di aprirlo viene restituito lo stesso codice d'errore, fino a quando nella directory DEVS:printers non viene inserito il driver di stampa selezionato. Si noti che il codice d'errore IOERR_OPENFAIL è assai generico, in quanto viene restituito anche quando il dispositivo non si trova nella directory DEVS:, quando non c'è abbastanza memoria nel sistema, e quando il dispositivo (che è ad accesso esclusivo) appartiene a un altro task. Per un task la causa d'errore più facile da diagnosticare è l'ultima. Infatti, cercando la struttura Device del dispositivo Printer nella lista di sistema DeviceList, e controllando il contenuto del parametro lib_OpenCnt può rendersi conto se il dispositivo è già in uso.

Questa stessa situazione si verifica anche quando il dispositivo Printer non riesce ad aprire Parallel o Serial per comunicare con la stampante, magari perché non li trova nella directory DEVS:, oppure perché non risulta abbastanza memoria libera nel sistema.

Se invece l'apertura del dispositivo ha successo, in memoria si troveranno il dispositivo Printer, il driver di stampa e il dispositivo Parallel (o Serial); il dispositivo Printer rimane in memoria fino a quando un task non decide di rimuoverlo tramite la funzione RemDevice (si noti che questo non provoca l'automatica rimozione dei dispositivi Parallel o Serial). Infine, occorre sottolineare che il driver di stampa presente in memoria rimane in uso anche se l'utente nel frattempo apre il tool Preferences e seleziona un altro driver. Il nuovo driver di stampa viene infatti caricato in memoria solo quando il dispositivo Printer viene chiuso e successivamente riaperto.

Funzionamento del dispositivo Printer

Il dispositivo Printer possiede un'unica unità. A essa vengono accodate tutte le richieste di elaborazione che i task inviano al dispositivo. La Figura 11.1 (nella pagina successiva) illustra il suo funzionamento generale.

Come abbiamo anticipato, esistono due modi per accedere alle routine interne del dispositivo Printer: da CLI (indirizzando il dispositivo logico PRT:) e tramite un task. In entrambi i casi, il testo da stampare può contenere solo codici ASCII e sequenze escape appartenenti a un sottoinsieme di quelle previste dallo standard ANSI X3.64. Questo standard, creato dall'American National Standards Institute, prevede un elevato numero di sequenze escape per la gestione dell'output di testi su video e stampanti. Il loro pregio maggiore è che consentono di svincolarsi dal particolare hardware di cui si dispone. Così come il dispositivo Console riconosce solo le sequenze ANSI che hanno efficacia sullo schermo, il dispositivo Printer prevede solo quelle che hanno efficacia su una stampante, come per esempio quelle che cambiano lo stile di scrittura, l'interlinea, le fonti-carattere, i margini... Un testo organizzato secondo questo

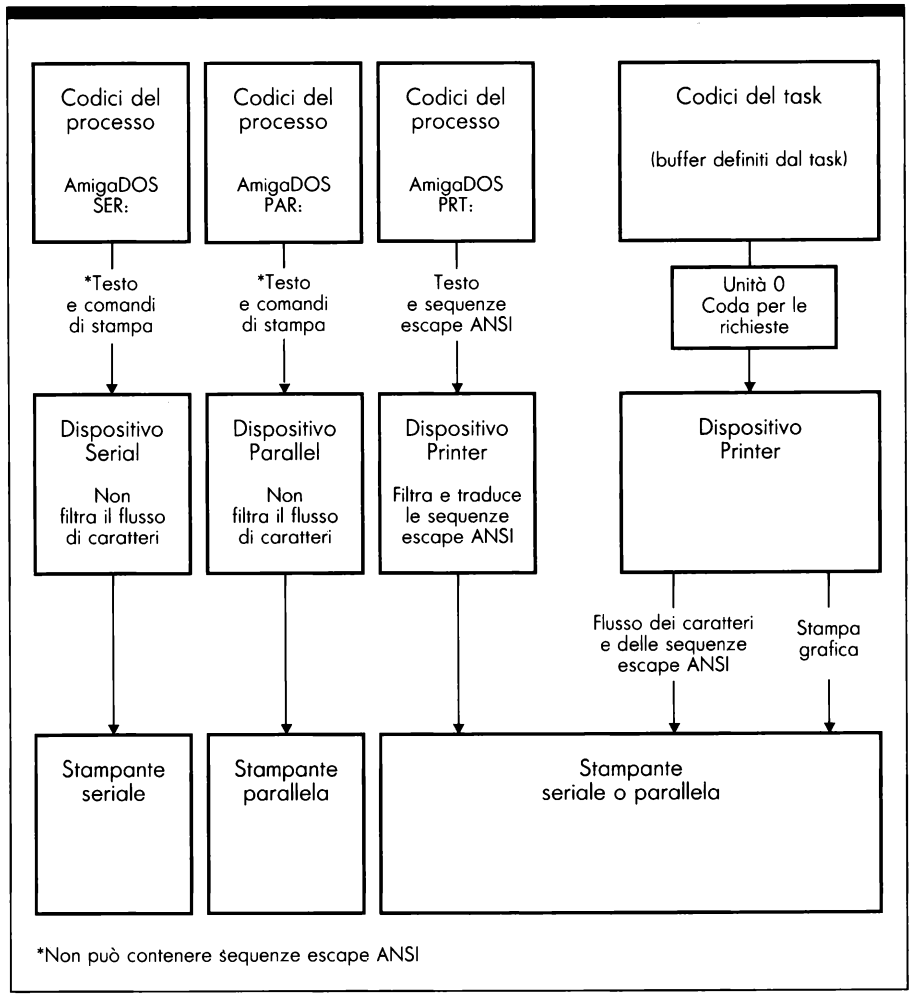


Figura 11.1: ***Funzionamento del dispositivo Printer***

standard prescinde completamente dal tipo di stampante e questo è il grande merito del dispositivo Printer: permette al software di essere relativamente indifferente al tipo di stampante collegata. Infatti, il compito del dispositivo Printer è proprio quello di elaborare un testo nel suddetto formato, traducendone tutte le sequenze escape ANSI X3.64 nelle sequenze escape previste dalla particolare stampante. Per compiere questa operazione deve ovviamente sapere quali comandi sono riconosciuti dalla stampante selezionata, e quali sono le sequenze escape che li attivano: il driver di stampa serve proprio per rispondere a queste domande.

Lo standard ANSI X3.64 previsto dal dispositivo Printer non deve comunque sembrare limitato perché non prevede tutte le possibili sequenze escape; infatti è prevista nello standard una particolare sequenza escape che

consente d'inviare codici direttamente alla stampante, evitando così che vengano filtrati dal parser del dispositivo Printer. Comunque, il dispositivo consente d'inviare alla stampante un testo da non filtrare anche con il comando PRD_RAWWRITE o con la funzione PWrite. In questo caso, l'unica differenza rispetto all'uso diretto del dispositivo Serial o Parallel riguarda una serie di sequenze escape d'inizializzazione che il dispositivo Printer invia alla stampante la prima volta che riceve un comando di scrittura. Questa particolare serie di sequenze escape varia a seconda del driver di stampa caricato in memoria. Il driver Generic non prevede sequenze d'inizializzazione, mentre per esempio il driver EpsonXOld prevede una sequenza che abilita il salto della perforazione, imposta l'interlinea a 1/6", disabilita il corsivo, il sottolineato, il modo enfatizzato, la doppia larghezza dei caratteri, i pedici e gli apici.

Il testo filtrato e trasformato dal dispositivo Printer viene infine inviato al dispositivo Parallel o Serial. Rispetto al testo originale, tutte le sequenze escape ANSI X3.64 dovrebbero essere state tradotte nelle sequenze escape riconosciute dalla stampante. Può capitare che per alcune la stampante non preveda un corrispondente comando; in questo caso la sequenza ANSI X3.64 viene semplicemente rimossa dal flusso senza essere sostituita con alcuna serie di codici. Quando il flusso di caratteri giunge alla stampante, il firmware in essa presente intercetta le sequenze escape ed esegue i relativi comandi. Le sequenze escape non vengono quindi stampate.

Il dispositivo Printer può essere aperto soltanto in modo esclusivo. Quando un task ne detiene il controllo nessun altro task può aprirlo a sua volta. È quindi molto importante chiuderlo sempre quando non è più necessario.

L'invio dei codici di controllo a una stampante

L'Amiga può inviare a una stampante qualsiasi carattere a 7 bit previsto dallo standard ANSI X3.64. Tutte le sequenze escape previste da questo standard iniziano con il carattere Escape (ASCII 27), seguito da alcuni caratteri diversi di volta in volta. L'Amiga prevede 75 sequenze escape per la stampa. Alcune di esse sono definite dall'Organizzazione Internazionale degli Standard (ISO, International Standards Organization), altre dalla Digital Equipment Corporation (DEC), altre ancora sono state create specificamente per l'Amiga.

Ogni stampante possiede il proprio insieme di codici di controllo. È compito di ciascun driver tradurre le sequenze escape ANSI X3.64 nei corrispondenti codici di controllo previsti dalla relativa stampante.

Per rendersi conto della grande versatilità del dispositivo Printer, consigliamo un semplicissimo esperimento.

1. Si seleziona il driver di stampa appropriato tramite il tool Preferences del Workbench. Questo tool può anche essere attivato da CLI digitando il comando PREFERENCES.

2. Si dirige l'input da tastiera verso la stampante attraverso il comando:

   COPY * TO PRT:

Questo comando provoca l'apertura di un file da tastiera che viene copiato nel dispositivo logico PRT:; questo file raccoglie i caratteri digitati dall'utente sulla tastiera.

3. Ora si deve digitare una sequenza escape. Per esempio attivare il corsivo premendo il tasto ESC e digitando la sequenza:

   [3m

   Questi tre caratteri, essendo preceduti dal carattere Escape, non appaiono sullo schermo, ma vengono ugualmente inseriti nel file da tastiera copiato nel dispositivo logico PRT:. Se ora si preme il Return, la riga digitata viene trasmessa al dispositivo PRT:. A questo punto, il dispositivo, insieme con il driver di stampa, analizza la riga e provvede a tradurre la sequenza escape nei corrispondenti codici di controllo previsti dalla stampante. Nel nostro esempio, alla stampante giunge il codice di controllo che abilita il corsivo.

4. Tenendo premuto il tasto Ctrl premere il tasto Backslash (\) per segnalare la fine del file da tastiera. Se la stampante collegata prevede la stampa in corsivo e se il driver di stampa prescelto non contiene errori, ogni successivo carattere inviato alla stampante viene stampato in corsivo (sempre che non contenga altre sequenze escape che alterino la configurazione raggiunta). Se, per esempio, si desidera stampare in corsivo un file il cui nome è MioFile, occorre impartire il comando:

   COPY MioFile TO PRT:

   oppure:

   COPY MioFile TO PAR:

   se non si desidera che il testo venga filtrato. Si noti che nell'intera procedura non è stato necessario ricorrere al manuale della stampante per ottenere la sequenza escape da essa prevista per l'abilitazione del corsivo. La procedura adottata vale per qualsiasi stampante connessa a una qualsiasi delle due porte (seriale o parallela). Proprio per questa ragione può essere facilmente standardizzata al fine d'inviare rapidamente particolari comandi alla stampante prima di stampare un testo agendo dal CLI.

Per automatizzare le procedure, si creano un certo numero di piccoli file contenenti sequenze escape ANSI X3.64 da inviare all'occorrenza. Modificando la startup-sequence, l'utente può fare in modo che questi file vengano caricati dal disco sistema nel RAM disk durante l'attivazione della macchina. Per esempio, è possibile:

- creare un file denominato ION per attivare la stampa in corsivo.

- Creare un file denominato IOFF per disattivare la stampa in corsivo.
- Creare un file denominato CON per attivare la stampa in modo condensato.
- Creare un file denominato COFF per disattivare la stampa in modo condensato.

Ognuno di questi file contiene una completa sequenza escape ANSI X3.64. Quindi, nel momento in cui si desidera attivare o disattivare una determinata funzione, non si fa altro che trasmettere l'appropriato file. Per esempio:

COPY ram:ION TO PRT:

per attivare la stampa in corsivo, oppure:

COPY ram:IOFF TO PRT:

per disattivarla.

comandi del dispositivo Printer

Il dispositivo Printer prevede quattro comandi specifici, una funzione e cinque comandi standard. Tutti i comandi, tranne PRD_PRTCOMMAND e CMD_WRITE, permettono il QuickIO. Tre comandi (CMD_FLUSH, CMD_START e CMD_STOP) prevedono anche il modo immediato. Tutti i comandi influenzano il parametro io_Error della struttura IOStdReq, IODRPReq oppure IOPrtCmdReq (queste strutture, come vedremo fra poco, occupano la stessa area di memoria RAM e possiedono gli stessi parametri fondamentali nei primi byte dell'area occupata; differiscono solo nei parametri non standard e la scelta dell'una o dell'altra dipende dal comando che si desidera inviare).

L'invio dei comandi al dispositivo Printer

Le Figure 11.2a e 11.2b (nelle pagine successive) descrivono lo schema generale utilizzato per inviare comandi al dispositivo Printer. Nella Figura 11.2a, le linee con le frecce rappresentano i parametri che occorre inizializzare, mentre nella Figura 11.2b rappresentano i parametri restituiti dalle routine interne del dispositivo Printer.

L'interazione con il dispositivo Printer prevede tre fasi.

1. *Preparazione della struttura di I/O*. In questa fase il programmatore ha il completo controllo; qui vengono inizializzati i parametri della struttura IOStdReq (oppure IOPrtCmdReq, o IODRPReq) in preparazione all'invio di un comando. La scelta della struttura e dei relativi parametri dipende

dal tipo di comando che il task intende impiegare.

2. *Invio della richiesta e sua elaborazione*. L'unico compito del programmatore in questa fase consiste nell'inviare il comando al dispositivo utilizzando le funzioni BeginIO, DoIO o SendIO. Il controllo passa successivamente alle routine interne del dispositivo che provvedono a elaborare la richiesta.

3. *Elaborazione dei parametri e loro restituzione*. Il sistema e le routine interne del dispositivo Printer controllano completamente questa fase. I risultati prodotti dall'elaborazione del comando vengono restituiti al task che ha inviato il comando. Se il QuickIO non ha avuto successo, la richiesta di I/O viene accodata alla reply port del task (dopo l'ascesa alla request port del dispositivo e la susseguente elaborazione). Se invece il

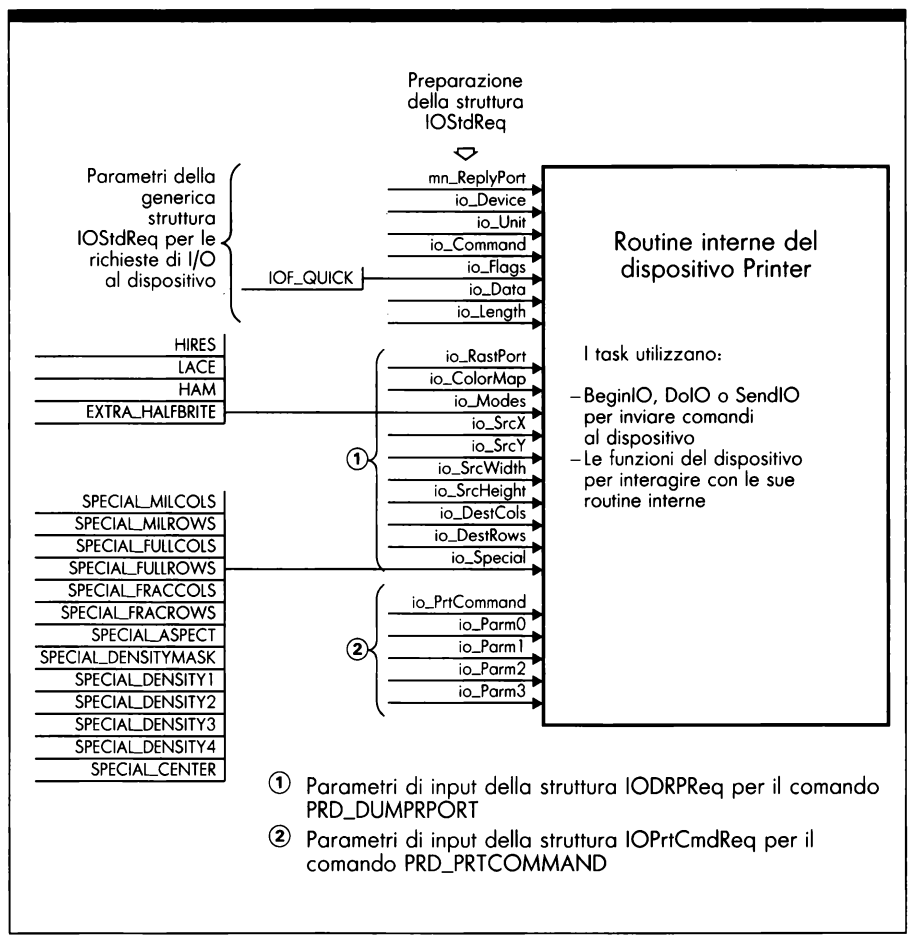


Figura 11.2a: *Gestione delle funzioni e dei comandi previsti dal dispositivo Printer (input)*

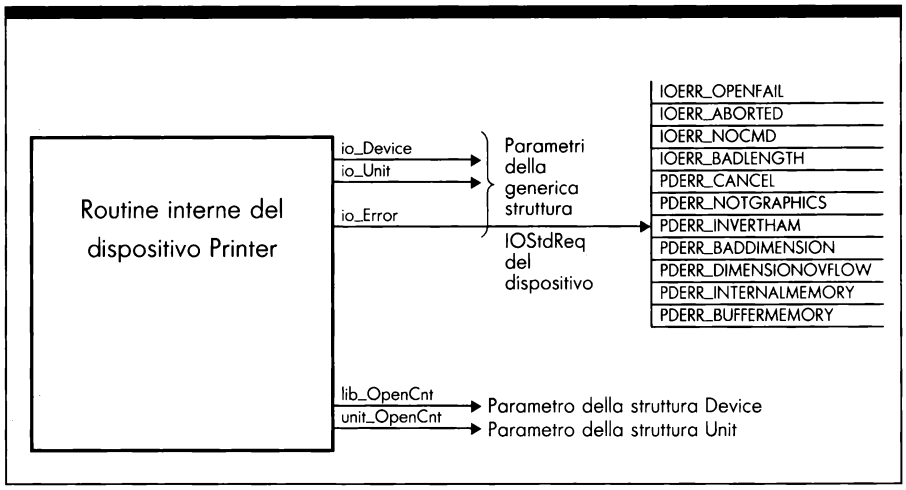


Figura 11.2b: *Gestione delle funzioni e dei comandi previsti dal dispositivo Printer (output)*

QuickIO ha avuto successo, la richiesta non viene accodata e la risposta torna direttamente al task.

Tutti i comandi restituiscono nel parametro io_Error un'informazione sul successo o sull'insuccesso ottenuto.

Le Figure 11.2a e 11.2b descrivono, inoltre, i parametri più significativi nella preparazione e nell'elaborazione delle funzioni del dispositivo Printer. OpenDevice e CloseDevice influenzano entrambe il parametro lib_OpenCnt della struttura Device; OpenDevice influenza anche il parametro io_Error.

Le procedure utilizzate per inviare i comandi alle routine interne del dispositivo Printer differiscono da quelle degli altri dispositivi principalmente per la necessità di tre diverse strutture di I/O. Si tratta di tre strutture che dal punto di vista della programmazione in C non conviene allocare in memoria separatamente. È improbabile che un task debba usarle contemporaneamente e quindi è più vantaggioso che occupino la stessa area di memoria. Nel linguaggio C questo è possibile tramite l'impiego di una "unione". Allocando l'unione ci si appropria di un'area di memoria sufficiente per contenere il più grande dei suoi elementi. Successivamente, è possibile indirizzare quest'area di memoria indicando il nome di uno dei tre elementi dell'unione. Nel caso del dispositivo Printer è consigliabile servirsi dell'unione PrinterIO, di cui parleremo nel prossimo paragrafo. Si tratta di un'unione non illustrata nei file INCLUDE, e quindi il programmatore deve definirla autonomamente all'interno del suo file sorgente.

L'unione PrinterIO

Quando un task desidera inviare un comando al dispositivo Printer, deve creare e inizializzare un'unione PrinterIO. Quest'operazione viene normalmente compiuta utilizzando la funzione CreateExtIO (di supporto alla libreria Exec) prima di chiamare la funzione OpenDevice. In questo modo, la stessa unione PrinterIO potrà essere utilizzata per tutti i comandi del dispositivo, anche se prevedono strutture di I/O fra loro differenti.

Si ricordi che, in linguaggio C, un'unione è in pratica un tipo di variabile che può contenere, in tempi diversi, dati di tipo diverso. Nell'unione PrinterIO, le strutture IOStdReq, IOPrtCmdReq e IODRPReq occupano tutte e tre i byte iniziali della stessa area RAM; il task può impiegare l'unione PrinterIO indicando una delle tre strutture senza che sia necessario allocare una diversa area RAM per ogni struttura. Questa organizzazione viene correttamente interpretata dal compilatore C e permette di risparmiare memoria.

I primi sei parametri di queste strutture (comuni a tutte le strutture per le richieste di I/O) sono perfettamente uguali. Le differenze fra le tre strutture riguardano il significato e il numero dei byte che seguono (cioè dei parametri successivi). La prima chiamata alla funzione OpenDevice, quella che per il task apre il dispositivo, inizializza i primi sei parametri qualunque sia la struttura impiegata. In seguito, quando viene fatta una scelta fra le tre strutture dell'unione, questi primi sei parametri risultano sempre correttamente inizializzati, e se uno di essi viene alterato, il nuovo valore che assume vale ovviamente per tutte e tre le strutture.

L'unione PrinterIO dev'essere definita come segue:

```
union PrinterIO {
        struct IOStdReq ios;
        struct IODRPReq iodrp;
        struct IOPrtCmdReq iopc;
} *PIO;
```

Si noti che con questa definizione vengono compiute due operazioni: viene creato un nuovo tipo di variabile, il tipo PrinterIO, e viene definita una variabile puntatore PIO che individua in memoria variabili di tipo PrinterIO. Il puntatore PIO è quello che poi dev'essere impiegato all'interno del file sorgente per individuare uno dei tre elementi contenuti nell'unione. Per creare in memoria un'unione di questo tipo, dopo aver definito l'unione PrinterIO si devono effettuare le seguenti inizializzazioni (esterne al programma):

```
struct MsgPort *CreatePort();
union PrinterIO *CreateExtIO();
struct MsgPort *PortaMsg;
```

e costruire il programma seguendo l'organizzazione illustrata nella pagina seguente.

```
main()
{
    PortaMsg = CreatePort("Mia Porta", ØL);
    PIO = CreateExtIO(PortaMsg, (long)sizeof(union PrinterIO));
    if (errore = OpenDevice("printer.device", ØL, PIO, ØL))
    {
        printf("Non posso aprire il dispositivo. Errore %ld\n", errore);
        DeletePort(PortaMsg);
        DeleteExtIO(PIO, (long)sizeof(union PrinterIO));
        exit(TRUE);
    };
    ...
}
```

Per leggere il contenuto del parametro lib_OpenCnt, il parametro io_RastPort della struttura IODRPReq e il parametro io_Parm0 della struttura IOPrtCmdReq occorre impartire le seguenti istruzioni:

```
printf("Contenuto di lib_OpenCnt %d\n",
    PIO->ios.io_Device->dd_Library.lib_OpenCnt);
printf("Contenuto di io_RastPort Øx%lx\n", PIO->iodrp.io_RastPort);
printf("Contenuto di io_ParmØ %d\n", PIO->iopc.io_ParmØ);
```

Osservando queste semplici istruzioni ci si può rendere conto di come dev'essere impiegata l'unione PrintIO per accedere ai parametri delle tre strutture che rappresenta.

La struttura IOStdReq costituisce la struttura di I/O che il task deve impiegare per tutti i comandi ad esclusione di PRD_DUMPRPORT e PRD_PRTCOMMAND. La struttura IODRPReq è richiesta solo per definire il comando non standard PRD_DUMPRPORT, mentre la struttura IOPrtCmdReq è richiesta solo per definire il comando non standard PRD_PRTCOMMAND. Quando nel parametro io_Command di una richiesta di I/O viene specificato uno di questi due comandi, il dispositivo si aspetta che siano stati preparati particolari parametri non standard, e soprattutto che questi parametri siano correttamente allocati come parte integrante della struttura di I/O. Il task deve avvalersi quindi della corrispondente struttura per inizializzarli correttamente. Gli altri parametri standard, i primi sei, si trovano impostati a valori di default, a meno che non siano stati alterati con l'invio di precedenti comandi. Si vedano le schede dei comandi per maggiori informazioni sui parametri che di volta in volta vanno presi in considerazione.

Le strutture del dispositivo Printer

Per definire e inviare i comandi del dispositivo Printer, occorre servirsi delle strutture IOStdReq, IOPrtCmdReq e IODRPReq. La struttura IOStdReq è stata presentata nel capitolo 2. La Figura 11.3 (nella pagina successiva) illustra

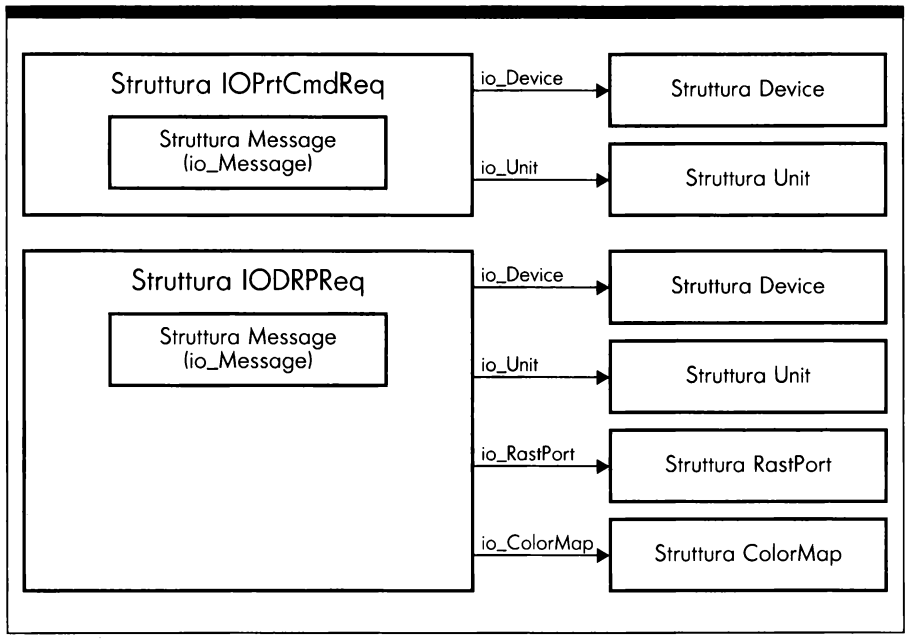


Figura 11.3: *Strutture utilizzate dal dispositivo Printer*

le strutture IOPrtCmdReq e IODRPReq, nelle quali i primi sei parametri sono gli stessi della struttura IOStdReq. Di essi diciamo solo che se si usa la struttura IOPrtCmdReq, il parametro io_Command deve contenere il comando PRD_PRTCOMMAND, mentre se si usa la struttura IODRPReq il parametro io_Command deve contenere il comando PRD_DUMPRPORT.

La struttura IOPrtCmdReq

La struttura IOPrtCmdReq viene utilizzata solo con il comando PRD_PRTCOMMAND, ed è definita come segue:

```
struct IOPrtCmdReq {
        struct Message io_Message;
        struct Device *io_Device;
        struct Unit *io_Unit;
        UWORD io_Command;
        UBYTE io_Flags;
        BYTE io_Error;
        UWORD io_PrtCommand;
        UBYTE io_ParmØ;
        UBYTE io_Parm1;
        UBYTE io_Parm2;
        UBYTE io_Parm3;
};
```

I suoi parametri hanno i seguenti significati:

- io_PrtCommand. Deve contenere il codice relativo alla sequenza escape ANSI X3.64 da inviare alla stampante. Nel file INCLUDE printer.h sono definite tutte le costanti corrispondenti alle sequenze escape ANSI X3.64 considerate dal dispositivo Printer. Per esempio, la sequenza escape che abilita il corsivo è ESC[3m; se il task desidera inviare alla stampante il comando di abilitazione del corsivo tramite il comando PRD_PRTCOMMAND, deve memorizzare nel parametro io_PrtCommand la costante aSGR3 (che vale 6), dal momento che il comando PRD_PRTCOMMAND interpreta questo valore come il comando che abilita il corsivo. PRD_PRTCOMMAND accede ai dati contenuti nel driver di stampa selezionato e verifica se la stampante prevede il corsivo. In caso affermativo, invia al dispositivo Serial o Parallel la sequenza di byte che per la stampante corrispondono al comando di abilitazione del corsivo. Se per esempio il driver di stampa prescelto è l'Epson XOld, alla stampante giungono i due caratteri 0x1B e 0x34, che nello standard Epson costituiscono il comando che abilita il corsivo. Qualora invece rilevi, consultando i dati contenuti nel driver di stampa, che per quella stampante non esiste un comando che abilita il corsivo (per esempio, se il driver di stampa è Generic), non invia alcun carattere e restituisce nel parametro io_Error il valore −1. Si noti che il comando PRD_PRTCOMMAND rappresenta l'unico caso in tutto il dispositivo Printer nel quale al posto delle sequenze escape ANSI X3.64 si devono usare le corrispondenti costanti. All'interno di un testo da inviare con il comando CMD_WRITE, per esempio, le sequenze escape ANSI X3.64 devono essere scritte per intero, così come sono previste dallo standard.

- io_Parm0, io_Parm1, io_Parm2 e io_Parm3. In questi parametri il task deve memorizzare sequenzialmente gli eventuali parametri previsti dalla sequenza escape ANSI X3.64. Per esempio, se si desidera indicare alla stampante una nuova lunghezza del foglio di stampa (espressa in numero di righe), occorre memorizzare nel parametro io_PrtCommand la costante aSLPP (il suo valore è 57), e nel parametro io_Parm0 il numero di righe che si preferisce, per esempio 66. Se il driver di stampa è l'Epson XOld, alla stampante giungeranno i byte 0x1B, 0x43 e 0x42, la codifica nello standard Epson del comando che imposta la lunghezza del foglio di stampa.

Per ottenere maggiori informazioni su questa struttura, si consulti il file INCLUDE denominato printer.h e la discussione relativa al comando PRD_PRTCOMMAND.

La struttura IODRPReq

La struttura IODRPReq viene utilizzata solo con il comando PRD_DUMPRPORT, che consente di stampare in grafica la copia di una bitmap a colori. Quella che segue è la sua definizione.

```
struct IODRPReq {
        struct Message io_Message;
        struct Device *io_Device;
        struct Unit *io_Unit;
        UWORD io_Command;
        UBYTE io_Flags;
        BYTE io_Error;
        struct RastPort *io_RastPort;
        struct ColorMap *io_ColorMap;
        ULONG io_Modes;
        UWORD io_SrcX;
        UWORD io_SrcY;
        UWORD io_SrcWidth;
        UWORD io_SrcHeight;
        LONG io_DestCols;
        LONG io_DestRows;
        UWORD io_Special;
};
```

I suoi parametri specifici hanno i seguenti significati.

- io_RastPort. Deve contenere l'indirizzo della struttura RastPort che definisce la bitmap per la quale si desidera ottenere la copia stampata (ovviamente in grafica).

- io_ColorMap. Deve contenere l'indirizzo della struttura ColorMap che definisce i colori relativi alla bitmap indicata dal parametro io_RastPort.

- io_Modes. Deve indicare in che modo vanno interpretati i dati presenti nella bitmap dell'immagine. Equivale al parametro Modes contenuto nella struttura ViewPort. Si tenga presente che la word più significativa di questo parametro è riservata, e dovrebbe essere sempre a zero.

- io_ScrX e io_ScrY. In questi parametri il task deve indicare l'offset orizzontale e verticale che dev'essere mantenuto dall'origine dell'immagine (raster bitmap) durante la stampa. L'origine è indicata nella struttura RastPort. Questi parametri servono, insieme ai due successivi, a stampare una bitmap parziale dell'intera bitmap.

- io_ScrWidth e io_ScrHeight. Questi parametri indicano la larghezza e l'altezza della bitmap che dev'essere inviata alla stampante. Sono

dimensioni rispettivamente relative ai parametri io_ScrX e io_ScrY. Insieme a questi due parametri, io_ScrWidth e io_ScrHeight permettono di stampare una bitmap parziale dell'intera bitmap.

- io_DestCols. Questo parametro ha a che fare con lo spazio orizzontale di stampa. Il suo significato dipende strettamente da come sono impostati i flag del parametro io_Special, come vedremo più avanti.

- io_DestRows. Questo parametro ha a che fare con lo spazio verticale di stampa. Il suo significato dipende strettamente da come sono impostati i flag del parametro io_Special, come vedremo descrivendo il comando PRD_DUMPRPORT.

- io_Special. È costituito da un insieme di flag che controllano l'interpretazione dei parametri io_DestCols e io_DestRows.

Si veda la discussione del comando PRD_DUMPRPORT e il file INCLUDE denominato printer.h per ottenere maggiori informazioni su questa struttura.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (printerIO)
A1

Scopo della funzione

Questa funzione chiude per il task l'accesso all'unità 0, l'unica prevista dal dispositivo Printer. All'apertura del dispositivo viene caricato in memoria il driver di stampa indicato dalla struttura Preferences e viene aperto il dispositivo Parallel o il dispositivo Serial, sempre seguendo le indicazioni contenute in Preferences. Chiudendo il dispositivo niente di tutto questo viene disallocato automaticamente. Se invece il task chiama la funzione RemDevice per eliminare il dispositivo dalla RAM, l'eliminazione rimane in sospeso fino a quando il task non esegue CloseDevice. In questo caso, quando CloseDevice

riceve infine il controllo, il dispositivo Printer e il driver di stampa vengono eliminati dalla memoria, mentre il dispositivo Parallel (o Serial) non viene toccato. L'eliminazione del dispositivo non crea problemi, in quanto essendo ad accesso esclusivo si è certi che nessun altro task lo sta impiegando. Quando la funzione CloseDevice è stata eseguita, il task non può più utilizzare il dispositivo Printer a meno che non chiami di nuovo la funzione OpenDevice per riaprirlo.

CloseDevice imposta a −1 i parametri io_Device e io_Unit della struttura di I/O. L'unione PrinterIO non può essere utilizzata nuovamente fino a quando questi parametri non verranno reinizializzati tramite una nuova chiamata a OpenDevice. CloseDevice decrementa inoltre il parametro lib_OpenCnt della struttura Device di gestione del dispositivo.

Argomenti della funzione

printerIO Deve contenere l'indirizzo dell'unione di tipo PrinterIO impiegata per interagire con il dispositivo.

Discussione

Un task dovrebbe sempre verificare che tutte le risposte alle richieste di I/O inviate siano state restituite prima di chiamare CloseDevice. Per effettuare questo controllo è possibile utilizzare le funzioni GetMsg, Remove, CheckIO e WaitIO.

Quando un task non ha più bisogno del dispositivo Printer, deve prontamente chiuderlo così che un altro task vi possa accedere.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("printer.device", ØL, printerIO, ØL)
DØ                 AØ                  DØ A1        D1
```

Scopo della funzione

Questa funzione apre per il task l'accesso all'unità 0, l'unica prevista dal dispositivo Printer. Se nella lista di sistema DeviceList non risulta la struttura Device di gestione di questo dispositivo, significa che non si trova in memoria. Il sistema provvede allora a caricarlo da disco in maniera trasparente al task. Quando è stato allocato in memoria, il sistema cede il controllo alla sua routine d'inizializzazione, la quale provvede a ad aprire il dispositivo Parallel o Serial (quello specificato nella struttura Preferences), e a caricare in memoria il driver di stampa indicato nella struttura Preferences. Se una di queste operazioni non ha successo, la funzione OpenDevice restituisce il codice d'errore IOERR_OPENFAIL. Le ragioni possono essere che il sistema non ha trovato nella directory logica DEVS: il dispositivo Parallel o Serial, oppure che nella directory DEVS:printers non ha trovato il driver di stampa indicato nella struttura Preferences. Infine, OpenDevice può anche non avere avuto successo perché il dispositivo Printer era occupato da un altro task; anche in questo caso viene restituito il codice d'errore IOERR_OPENFAIL.

Una volta che OpenDevice ha aperto il dispositivo Printer, ne inizializza i parametri interni e memorizza nei parametri io_Device e io_Unit della struttura di I/O indicata dal task gli indirizzi delle strutture Device e Unit. Inoltre incrementa il parametro lib_OpenCnt della struttura Device di gestione del dispositivo. Si noti che i parametri io_Device e io_Unit sono contenuti in tutte e tre le strutture rappresentate dall'unione di tipo PrinterIO. OpenDevice richiede una reply port inizializzata in modo adeguato, e un eventuale bit di segnalazione allocato a quella porta se il task desidera essere avvertito quando le routine del dispositivo Printer restituiscono una risposta. I risultati prodotti dall'esecuzione della funzione sono restituiti nei parametri che seguono.

- io_Device. Contiene l'indirizzo della strutture Device di gestione del dispositivo. Si noti che in realtà questo indirizzo individua in memoria una struttura più complessa della struttura Device, la struttura PrinterData, definita nel file INCLUDE prtbase.h. Si tratta di una struttura molto più estesa, che contiene però una struttura Device come primo parametro. In particolare, il parametro pd_Preferences è una struttura Preferences che contiene la copia di tutti i parametri della struttura Preferences di sistema. Se per esempio un task vuole sapere qual è il driver di stampa selezionato, può accedere all'array:

```
((struct PrinterData *)PIO->ios.io_Device)
                    ->pd_Preferences.PrinterFilename
```

  e leggere la stringa in esso contenuta.

- io_Unit. Contiene l'indirizzo della struttura Unit di gestione dell'unità 0, l'unica prevista dal dispositivo Printer. Al suo interno la struttura MsgPort definisce la request port dell'unità e la relativa coda.

- io_Error. Il valore 0 indica che la richiesta è stata eseguita. IOERR_OPENFAIL indica che il dispositivo Printer non è stato aperto; questo può significare che il dispositivo Printer non si trova in memoria e non risulta su disco, che è già in uso da un altro task, che il driver di stampa prescelto non è presente nel disco, che non è presente né in memoria né su disco il dispositivo Parallel (o Serial), o infine che nel sistema non c'è abbastanza memoria.

Argomenti della funzione

"printer.device"	Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo Printer.
ØL	Indica che si desidera aprire l'unità 0 del dispositivo, l'unica disponibile nel dispositivo Printer.
printerIO	Deve contenere l'indirizzo dell'unione di tipo PrinterIO da impiegare come struttura di I/O.
ØL	Indica che la funzione ignora l'argomento flag.

Preparazione dell'unione PrinterIO

Per aprire il dispositivo Printer occorre inizializzare il parametro mn_ReplyPort della struttura di I/O con l'indirizzo della struttura MsgPort rappresentante la reply port del task. Il task può allocare una message port tramite la funzione CreatePort di supporto alla libreria Exec, e indicarne l'indirizzo come argomento della funzione CreateExtIO, sempre di supporto alla libreria Exec. Chiamando quest'ultima funzione il task alloca la struttura di I/O necessaria per interagire con il dispositivo e memorizza automaticamente l'indirizzo della reply port nel parametro mn_ReplyPort.

Discussione

OpenDevice permette di aprire l'unità 0 del dispositivo Printer. Trattandosi di un'unità ad accesso esclusivo, ed essendo l'unica prevista dal dispositivo, se la funzione OpenDevice ha successo il task detiene il controllo dell'intero dispositivo, e nessun altro task può accedervi fino a quando non viene chiamata la funzione CloseDevice. Una volta che il dispositivo è stato aperto, il task può inviare una serie di comandi alla stampante, la quale dev'essere connessa alla porta indicata nella struttura Preferences di sistema. L'utente può modificare questa struttura (e quindi alterare i parametri che intervengono nella stampa)

tramite il tool Preferences. Per esempio, può selezionare un nuovo driver di stampa, può indicare che la stampante è collegata alla porta parallela piuttosto che a quella seriale e così via.

Quando un task non ha più bisogno del dispositivo Printer, deve provvedere a chiuderlo tramite la funzione CloseDevice, in modo che altri task possano servirsene.

PWrite

Sintassi di chiamata della funzione

```
error = (*((struct PrinterData *)printerIO->ios.io_Device)
                ->pd_PWrite)(Stringa, Lunghezza)
```

Il dispositivo prevede che la struttura puntata da io_Device sia di tipo PrinterData (viene definita nel file INCLUDE prtbase.h); come si ricorderà, per il dispositivo Printer io_Device individua in memoria una struttura di tipo PrinterData, e non Device. Il puntatore printerIO deve individuare in memoria l'unione di tipo PrinterIO allocata dal task e inizializzata tramite OpenDevice. Infine, pd_PWrite è un puntatore della struttura PrinterData che individua in memoria la funzione PWrite. Se all'interno del task questa funzione viene chiamata con una certa frequenza, oppure capita spesso di indirizzare parametri della struttura PrinterData, è più conveniente definire un puntatore nel modo seguente:

```
PD = (struct PrinterData *)printerIO->ios.io_Device;
```

e chiamare la funzione PWrite con la seguente istruzione:

```
error = (*PD->pd_PWrite)(Stringa, Lunghezza);
```

Scopo della funzione

PWrite è una particolare funzione che consente d'inviare alla stampante una stringa di byte aggirando il parser interno del dispositivo. Generalmente questa funzione viene impiegata dal dispositivo stesso per inviare alla stampante il contenuto del proprio buffer interno, ma dal momento che il suo indirizzo è contenuto nel parametro pd_PWrite della struttura PrinterData individuata da io_Device, anche i task all'occorrenza possono impiegarla.

Il contenuto della stringa viene analizzato soltanto quando nell'argomento Lunghezza il task indica il valore −1. In questo caso, infatti, la funzione invia

alla stampante tutti i byte che incontra nella stringa fino a quando non incontra un byte a 0 (che viene anch'esso inviato). Con questo metodo, in pratica, viene inviata una stringa a terminazione nulla, ed è l'unico caso nel quale la funzione esamina il contenuto della stringa.

Se invece il task indica nell'argomento Lunghezza un valore diverso da −1, la funzione lo interpreta come il numero di caratteri che deve leggere dalla stringa e inviare alla stampante; gli eventuali byte a zero contenuti nella stringa vengono semplicemente trasmessi.

La funzione restituisce il valore 0 nella variabile error se l'operazione ha avuto successo. Un valore diverso da zero, invece, significa che si è verificata una condizione d'errore.

Argomenti della funzione

Stringa Deve contenere l'indirizzo della stringa di byte che si desidera inviare alla stampante. La stringa può contenere byte di qualunque valore, dal momento che la funzione li invia direttamente, senza esaminarli. Occorre solo ricordare che se l'argomento Lunghezza viene impostato a −1, il primo byte a zero presente nella stringa determina la fine della trasmissione.

Lunghezza Deve contenere la lunghezza della stringa da trasmettere alla stampante (si ricordi che la funzione si aspetta in questo parametro una long word). Se il task indica il valore −1, la funzione trasmette i byte contenuti nella stringa fino a quando non incontra un byte a zero (che viene trasmesso): in pratica viene inviata una stringa a terminazione nulla.

Discussione

Tramite questa speciale funzione di basso livello, i task hanno a disposizione un metodo molto rapido per inviare una stringa di byte alla stampante senza che intervenga alcun filtro sul suo contenuto. Nella stringa possono quindi essere presenti codici di controllo di qualunque formato. Impiegando PWrite, l'unico vantaggio rispetto all'uso diretto del dispositivo Parallel (o Serial) è che tutti i parametri della trasmissione sono gestiti dal dispositivo Printer in maniera trasparente al task, il quale non ha quindi necessità di sapere a quale porta è collegata la stampante o a quale velocità dev'essere effettuata la trasmissione se il collegamento è seriale. Si noti che la chiamata alla funzione PWrite è piuttosto atipica, in quanto non prevede l'impiego da parte del task di una struttura di I/O.

COMANDI STANDARD DEL DISPOSITIVO

CMD_FLUSH

Scopo del comando

CMD_FLUSH elimina tutte le richieste di I/O accodate e in esecuzione. Viene eseguito in modo immediato, permette il QuickIO, e viene restituito alla reply port del task soltanto se il QuickIO non è stato richiesto. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato io_Command in modo non corretto.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_FLUSH e inizializzare io_Flags a IOF_QUICK se intende richiedere il QuickIO; altrimenti deve inizializzarlo a 0.

Discussione

CMD_FLUSH elimina tutte le richieste PRD_DUMPRPORT, PRD_PRTCOMMAND, PRD_RAWWRITE oppure CMD_WRITE, sia quelle in esecuzione sia quelle accodate alla request port dell'unità. Dato che CMD_FLUSH ha effetti distruttivi, si deve utilizzarlo con estrema prudenza. Ogni richiesta di I/O eliminata viene restituita al mittente con il codice d'errore IOERR_ABORTED nel parametro io_Error.

CMD_RESET

Scopo del comando

CMD_RESET reinizializza l'unità 0 del dispositivo Printer, riportandola allo stato in cui si trovava al momento dell'apertura del dispositivo. CMD_RESET permette il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non è stato richiesto. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato io_Command in modo non corretto.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_RESET e inizializzare io_Flags a IOF_QUICK se intende richiedere il QuickIO; altrimenti deve inizializzarlo a 0.

Discussione

CMD_RESET inizializza nuovamente il dispositivo, riportandolo alle condizioni di default.

CMD_START

Scopo del comando

CMD_START riattiva l'unità 0 del dispositivo Printer, se era stata bloccata tramite il comando CMD_STOP. La riattivazione riguarda ovviamente anche l'elaborazione in atto. CMD_START è un comando immediato, permette il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non è stato richiesto. L'esito del comando viene restituito nel parametro io_Error. Il

valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato io_Command in modo non corretto.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_START e inizializzare io_Flags a IOF_QUICK se intende richiedere il QuickIO; altrimenti deve inizializzarlo a 0.

Discussione

CMD_START è simile al comando Ctrl-Q utilizzato per far ripartire la visualizzazione dell'output sulla maggior parte dei computer. Esso riattiva immediatamente l'esecuzione di un comando bloccato tramite CMD_STOP, così come Ctrl-Q fa riprendere la visualizzazione dell'output precedentemente bloccata con Ctrl-S. CMD_START fa anche riprendere l'ascesa nella coda alla request port delle richieste di I/O accodate.

CMD_STOP

Scopo del comando

CMD_STOP blocca l'esecuzione di un comando e impedisce che le routine interne del dispositivo Printer elaborino le richieste di I/O accodate. Anche se l'unità 0 è stata bloccata con CMD_STOP, il sistema continua ad accodare le richieste inviate all'unità. Il dispositivo Printer, comunque, può elaborare soltanto richieste di tipo CMD_START o CMD_FLUSH.

CMD_STOP viene eseguito in modo immediato, permette il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non è stato richiesto. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il task ha specificato io_Command in modo non corretto.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_STOP e inizializzare io_Flags a IOF_QUICK se intende richiedere il QuickIO; altrimenti deve inizializzarlo a 0.

Discussione

CMD_STOP blocca l'esecuzione dei comandi PRD_DUMPRPORT, PRD_PRTCOMMAND, PRD_RAWWRITE o CMD_WRITE. La sua funzione è simile a quella del comando Ctrl-S utilizzato per bloccare la visualizzazione dell'output su schermo nella maggior parte dei computer. Le richieste di I/O presenti nella coda alla request port del dispositivo rimangono bloccate finché non arriva un comando CMD_START.

CMD_WRITE

Scopo del comando

Il comando CMD_WRITE provoca il trasferimento di un flusso di caratteri in standard ANSI X3.64 dal buffer del task alla stampante. Il dispositivo Printer "filtra" il flusso, trasformando nei codici di controllo previsti dalla particolare stampante tutte le sequenze escape che incontra; il flusso risultante viene poi inviato al dispositivo Parallel (o Serial). La trasformazione delle sequenze escape ANSI X3.64 avviene secondo le informazioni contenute nel driver di stampa che il dispositivo ha caricato in memoria. Se una sequenza escape ANSI X3.64 non corrisponde ad alcun comando della stampante, oppure se non rientra fra quelle riconosciute dal dispositivo, la sequenza viene semplicemente rimossa dal flusso senza che il task ne venga avvertito. È quindi importante che un task verifichi se la stampante prevede un comando per ognuna delle sequenze escape che intende usare. Per operare questo controllo può servirsi del comando PRD_PRTCOMMAND, che restituisce un errore se una particolare sequenza non corrisponde a nessun comando della stampante.

Il numero di caratteri da trasferire dev'essere indicato nel parametro io_Length della richiesta di I/O. Se viene specificato il valore −1, il comando CMD_WRITE continua a trasferire caratteri fino a quando non incontra un byte a 0, che viene trasmesso come ultimo carattere.

Dato che il comando CMD_WRITE non permette il QuickIO, le richieste di

I/O vengono sempre restituite alla reply port del task che le aveva inviate. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. PDERR_INTERNALMEMORY indica che non c'era sufficiente memoria libera per le routine interne del dispositivo Printer nel momento in cui è stato inviato il comando CMD_WRITE. PDERR_BUFFERMEMORY indica che non c'era sufficiente memoria libera per contenere il buffer definito dal task. IOERR_NOCMD indica che il task ha specificato il parametro io_Command in modo non corretto e IOERR_BADLENGTH indica che il task ha specificato in modo non corretto il parametro io_Length.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando CMD_WRITE.

Deve inoltre inizializzare io_Length con il numero dei caratteri che intende inviare alla porta seriale o parallela, oppure inizializzarlo a −1 per indicare che il testo è una stringa a terminazione nulla e dev'essere inviato fino al byte zero compreso. Infine deve inizializzare io_Data con l'indirizzo della stringa.

Vediamo un esempio d'inizializzazione dei parametri per inviare un testo che abilita il corsivo e stampa la frase "Corsivo abilitato".

```
printerIO->ios.io_Command = CMD_WRITE;
printerIO->ios.io_Length = -1L;
printerIO->ios.io_Data = (APTR)"\x1b[3mCorsivo abilitato";
SendIO(printerIO); /* Cosi' il task non entra in pericolose attese */
```

Si noti all'interno della stringa a terminazione nulla la presenza della sequenza escape ANSI X3.64 che attiva il corsivo, ricordando che i caratteri "\x1b" all'interno di una stringa in linguaggio C indicano al compilatore che in quel punto dev'essere inserito un byte contenente il valore esadecimale 0x1B. Il flusso di caratteri che giungono effettivamente alla stampante dipende da quale driver di stampa è in uso. Se il driver di stampa dell'esempio fosse Generic, l'intera sequenza escape verrebbe ignorata, e alla stampante giungerebbe la stringa "Corsivo abilitato\0" (si noti che lo zero viene inviato automaticamente come terminatore della stringa: se si desidera evitarlo occorre indicare l'esatta lunghezza della stringa). Se invece il driver di stampa fosse per esempio Epson XOld, prima della stringa "Corsivo abilitato\0" giungono alla stampante i due caratteri 0x1B e 0x34, che nello standard Epson corrispondono al comando che abilita il corsivo.

Discussione

Il dispositivo Printer adatta il flusso di caratteri alle specifiche della stampante collegata: tutte le sequenze escape ANSI X3.64 presenti nel flusso vengono sostituite da codici corrispondenti ai comandi della stampante. Il dispositivo Printer accede al driver di stampa indicato dal task per sapere se esiste un comando della stampante associato alla particolare sequenza escape ANSI.

Se si desidera inviare alla stampante un codice di controllo per il quale il dispositivo non prevede sequenze escape nello standard ANSI X3.64, si possono scegliere due strade. Si può inviare la particolare sequenza di byte alla stampante tramite il comando PRD_RAWWRITE o tramite la funzione PWrite, con lo svantaggio però di dover spezzare il testo da trasmettere in due blocchi e inviare la serie di byte non standard con un comando diverso da CMD_WRITE; questa procedura talvolta è quasi impraticabile. Il secondo metodo consiste nell'impiegare all'interno del testo la sequenza escape ESC[Pn, la quale indica al dispositivo che gli n caratteri che seguono non devono essere filtrati dal parser, cioè devono essere inviati direttamente così come sono. Si noti che per inviare una singola sequenza escape ANSI X3.64 non contenuta all'interno di un testo, conviene invece impiegare il comando PRD_PRTCOMMAND e le costanti definite nel file INCLUDE prtbase.h.

COMANDI SPECIFICI DEL DISPOSITIVO

PRD_DUMPRPORT

Scopo del comando

Il comando PRD_DUMPRPORT provoca la stampa parziale o totale di una bitmap (un'immagine). Per specificare il modo in cui dev'essere stampata, vengono utilizzati diversi parametri delle strutture RastPort, ViewPort, ColorMap. Inoltre, i parametri della struttura IODRPReq (la struttura non standard da usare con questo comando) vengono utilizzati per controllare la grandezza e altre caratteristiche che dovrà possedere la copia su carta dell'immagine (si tratta ovviamente di una stampa in grafica).

PRD_DUMPRPORT permette il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non è stato richiesto. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. Gli altri possibili errori sono i seguenti:

- PDERR_NOTGRAPHICS indica che la stampante descritta dal driver di stampa non è adatta al modo grafico e non può quindi stampare la bitmap.

- PDERR_BADDIMENSION indica che le dimensioni specificate nella struttura IODRPReq non sono accettabili.

- PDERR_DIMENSIONOVFLOW indica che le dimensioni specificate nella struttura IODRPReq hanno provocato un errore di overflow.

- IOERR_NOCMD indica che il task ha specificato il parametro io_Command in modo non corretto.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando PRD_DUMPRPORT e inizializzare io_Flags a IOF_QUICK se intende richiedere il QuickIO; altrimenti deve inizializzarlo a 0.

Deve inoltre inizializzare i seguenti parametri della struttura IODRPReq (si veda il capitolo 2 del volume I per una dettagliata descrizione delle strutture RastPort e BitMap qui citate):

- io_RastPort. Deve contenere l'indirizzo della struttura RastPort che descrive la bitmap da stampare.

- io_ColorMap. Deve contenere l'indirizzo della struttura ColorMap che descrive i colori della bitmap.

- io_Modes. Deve contenere il valore relativo al modo di stampa desiderato, cioè il modo in cui verrà interpretata la bitmap. Il significato di questo parametro è lo stesso del parametro Modes della struttura ViewPort; si tenga presente che la word più significativa di questo parametro è riservata.

- io_ScrX. Deve contenere l'offset orizzontale dall'origine della bitmap.

- io_ScrY. Deve contenere l'offset verticale dall'origine della bitmap.

- io_ScrWidth. Deve contenere la larghezza, riferita all'origine (io_ScrX, io_ScrY), della bitmap parziale da stampare.

- io_ScrHeight. Deve contenere l'altezza, riferita all'origine (io_ScrX, io_ScrY), della bitmap parziale da stampare.

- io_DestCols. Deve contenere un valore coerente con il significato assegnatogli dal parametro io_Special. In generale ha a che fare con la lunghezza orizzontale di stampa.

- io_DestRows. Deve contenere un valore coerente con il significato assegnatogli dal parametro io_Special. In generale ha a che fare con la lunghezza verticale di stampa.

- io_Special. Questo parametro è costituito da una serie di flag. Se s'impostano i flag SPECIAL_MILCOLS e SPECIAL_MILROWS, i valori indicati nei parametri io_DestCols e io_DestRows devono essere espressi in millesimi di pollice; per esempio, io_DestCols = 8.000 e io_DestRows = 10.500 provocheranno una stampa di 8 x 10,5 pollici. Se s'impostano i flag SPECIAL_FULLCOLS e SPECIAL_FULLROWS, i valori indicati nei parametri io_DestCols e io_DestRows vengono ignorati; in questo caso la bitmap viene stampata nelle dimensioni massime della pagina che la stampante è in grado di gestire, oppure secondo i limiti imposti dalla configurazione di sistema. Se s'impostano i flag SPECIAL_FRACCOLS e SPECIAL_FRACROWS, i valori indicati nei parametri io_DestCols e io_DestRows individuano una frazione della pagina stampabile nelle due direzioni X e Y; in questo caso, i parametri sono intesi come frazioni, espresse su long word (32 bit), delle massime dimensioni della pagina stampabile. Se s'imposta il flag SPECIAL_CENTER l'immagine viene stampata centrata fra il margine sinistro e destro del foglio di stampa. Se s'imposta il flag SPECIAL_ASPECT, una delle due dimensioni io_DestCols e io_DestRows può essere aumentata o diminuita per garantire che l'immagine stampata mantenga le giuste porporzioni anche se la stampante ha un aspect ratio diverso da 1 (l'altezza e la larghezza del punto di stampa differiscono di un fattore costante). L'operazione compiuta dal dispositivo quando s'imposta il flag SPECIAL_ASPECT si chiama scale (o scaling), e prevede l'impiego di un fattore di scala per modificare una coordinata. Questa opzione prende in considerazione i dati riportati nei precedenti sei parametri. Si noti che se tutti i flag SPECIAL_MIL/FULL/FRAC e SPECIAL_ASPECT sono a zero, i valori indicati nei parametri io_DestCols e io_DestRows vengono considerati espressi in pixel.

  I sette flag SPECIAL_DENSITY1/2/3/4/5/6/7 servono per variare la densità di stampa. SPECIAL_DENSITY1 corrisponde alla densità più bassa. Impostare il flag SPECIAL_NOFORMFEED se si desidera che al termine della stampa della bitmap non venga inviato il comando di salto pagina; questo comando è utile quando la stampante è di tipo page-oriented (come le stampanti laser) e si desidera stampare grafica e testo sullo stesso foglio. Impostando il flag SPECIAL_TRUSTME, il driver di stampa non invia alla stampante una sequenza di reset prima di cominciare la stampa della bitmap; alcuni driver di stampa, però, non prendono in considerazione questo flag. Comunque, le ultime disposizioni ufficiali per la creazione dei driver di stampa richiedono ai programmatori di tenerne conto, e quindi è consigliabile che il task lo

imposti. Si deve impostare il flag SPECIAL_NOPRINT se si desidera che vengano impostati i parametri di stampa, ma che la stampa non venga effettuata. Se il task invia la richiesta di I/O con questo flag impostato, può poi accedere ai parametri io_DestRows e io_DestCols per rilevare quali sono le condizioni di stampa.

Se tutti i flag di io_Special vengono lasciati a zero, i parametri io_DestCols e io_DestRows vengono interpretati come larghezza e altezza (espresse in punti di stampa) dell'immagine stampata.

Discussione

Il comando PRD_DUMPRPORT permette a un task di stampare una bitmap intera o parziale. La stampa può essere a colori oppure a scala di grigi. Si ricordi che una bitmap viene completamente descritta dalle strutture RastPort, ViewPort, ColorMap e BitMap della libreria Graphics (si veda il capitolo 2 del volume I). Un task, esattamente come usa i parametri di queste strutture per definire le modalità di visualizzazione di una bitmap, può utilizzarli anche per definire la stampa parziale o totale di una bitmap.

Per esempio, se si desidera creare uno stampato che illustri il funzionamento di un programma che utilizza Intuition, si può stampare uno schermo Workbench contenente una delle sue finestre. Per compiere tale operazione, occorre aprire la libreria Intuition (con la funzione OpenLibrary); inizializzare la struttura NewWindow; aprire una finestra (con la funzione OpenWindow); copiare a parte il puntatore alla struttura ViewPort, e i parametri delle strutture RastPort, ViewPort, ColorMap e BitMap che definiscono lo schermo Workbench. Questi parametri si possono poi utilizzare per definire quelli della struttura IODRPReq utilizzata per inviare il comando PRD_DUMPRPORT. Eseguite queste operazioni di copia, si dà il via alla stampa dello schermo del Workbench inviando il comando PRD_DUMPRPORT.

PRD_PRTCOMMAND

Scopo del comando

Il comando PRD_PRTCOMMAND provoca l'invio alla stampante di una singola sequenza escape ANSI X3.64. A differenza del comando CMD_WRITE, con PRD_PRTCOMMAND la sequenza da inviare non si indica per esteso, cioè sotto forma di stringa ASCII, ma tramite una particolare costante a essa associata. Nel file INCLUDE prtbase.h viene associata una costante a ognuna delle sequenze escape ANSI X3.64 riconosciute dal dispositivo Printer. Se si volesse per esempio attivare il neretto tramite il comando

PRD_PRTCOMMAND, si dovrebbe indicare la costante aSGR1 (di valore 10), che il comando interpreta come l'arrivo della sequenza escape ESC[1m.

Per inviare il comando PRD_PRTCOMMAND, il task deve impiegare una struttura di tipo IOPrtCmdReq, nella quale dopo i primi sei parametri standard seguono il parametro io_PrtCommand (che contiene il codice della sequenza escape da inviare) e altri quattro parametri, io_Parm0/1/2/3 nei quali il task deve indicare gli eventuali parametri richiesti dalla sequenza escape. Una volta che il comando è stato inviato, le routine interne del dispositivo Printer accedono al driver di stampa presente in memoria per sapere se la stampante prevede un comando per la sequenza escape. Se il comando esiste lo inviano alla stampante, mentre se non esiste restituiscono al task il codice d'errore −1 nel parametro io_Error della struttura di I/O. Questa procedura può servire a un task per controllare se il driver di stampa, e quindi la stampante, prevede un comando per una particolare sequenza escape ANSI X3.64. Se per esempio si invia il codice ANSI aSGR3 (di valore 6, corrispondente alla sequenza che abilita il corsivo) quando il driver di stampa è Generic, alla stampante non giunge alcun comando e il task ottiene il codice d'errore −1.

Dato che PRD_PRTCOMMAND non permette il QuickIO, le sue strutture di I/O vengono sempre restituite alla reply port del task che ha inviato il comando. L'esito del comando viene restituito nel parametro io_Error. A parte il valore −1 sopra descritto, il valore 0 indica che il comando è stato eseguito; PDERR_INTERNALMEMORY indica che non c'è abbastanza memoria per le routine del dispositivo al momento dell'invio del comando PRD_PRTCOMMAND; IOERR_NOCMD indica che il task ha specificato il parametro io_Command in modo non corretto.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando PRD_PRTCOMMAND e inizializzare io_Flags a 0.

Nel parametro printerIO−>iopc.io_PrtCommand il task deve memorizzare il codice corrispondente alla sequenza escape ANSI X3.64 che desidera inviare al dispositivo. Questi codici sono definiti nel file INCLUDE prtbase.h. Nei successivi parametri della struttura IOPrtCmdReq il task deve sequenzialmente indicare quelli eventualmente previsti dalla sequenza escape. I parametri che non vengono impiegati devono essere azzerati.

Discussione

PRD_PRTCOMMAND permette a un task d'inviare una singola sequenza escape ANSI X3.64 alla stampante. Rispetto al comando CMD_WRITE ha il

pregio di essere più rapido da definire e da eseguire, e soprattutto di segnalare se alla sequenza escape indicata non corrisponde alcun codice della stampante.

PRD_QUERY

Scopo del comando

Questo comando restituisce lo stato delle linee che giungono alla porta a cui è collegata la stampante. Dal momento che il dispositivo Printer impiega la porta parallela o quella seriale, la "parola di stato" restituita descrive lo stato della connessione parallela o seriale. Il comando restituisce nel parametro io_Actual il valore 1 per indicare che è in uso il dispositivo Parallel, o il valore 2 per indicare che è in uso il dispositivo Serial. Il task, sulla base di questo valore, deve interpretare nel giusto modo i dati riportati nella word puntata dal parametro io_Data.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve inizializzare io_Command con il comando PRD_QUERY; inizializzare io_Flags a 0 e memorizzare nel parametro io_Data l'indirizzo di una word che ha allocato in memoria e nella quale desidera sia riportato lo stato del collegamento.

Discussione

Quando il comando viene restituito, il task deve accedere al parametro io_Actual per rilevare se il collegamento con la stampante avviene tramite la porta parallela (io_Actual = 1) o quella seriale (io_Actual = 2). Successivamente deve accedere ai bit della word puntata da io_Data per sapere qual è lo stato del collegamento. Se è in uso la porta parallela i bit devono essere interpretati nel modo seguente: bit 0 azzerato, stampante off-line; bit 1 azzerato, mancanza di carta; bit 2 azzerato, stampante on-line (si osservi che la linea elettrica corrispondente a questo bit giunge anche alla porta seriale, nella quale rileva la presenza di una chiamata, principalmente nelle comunicazioni via modem).

Se invece è in uso la porta seriale, i bit della word puntata da io_Data devono essere interpretati nel modo seguente: bit 2 impostato, Ring Indicator (indicatore di chiamata); bit 3 azzerato, Data Set Ready (dato pronto); bit 4

azzerato, Clear To Send (azzera per trasmettere); bit 5 azzerato, Carrier Detected (portante presente); bit 6 azzerato, Ready To Send (pronto per trasmettere); bit 7 azzerato, Data Terminal Ready; bit 8 impostato, overflow del buffer di lettura del dispositivo Serial; bit 9 impostato, segnale di break inviato; bit 10 impostato, segnale di break ricevuto; bit 11 impostato, trasmesso XOFF; bit 12 impostato, ricevuto XOFF.

PRD_RAWWRITE

Scopo del comando

Il comando PRD_RAWWRITE provoca il trasferimento dal buffer del task alla stampante di un flusso di caratteri senza che venga compiuta alcuna analisi su di essi, cioè senza che intervenga il parser del dispositivo, come invece accade con il comando CMD_WRITE. Generalmente questi caratteri consistono di sequenze escape dedicate alla particolare stampante e non previste dallo standard ANSI X3.64.

Il task deve indicare nel parametro io_Length della richiesta il numero di caratteri da trasferire. Contrariamente a quanto accade per il comando CMD_WRITE e per la funzione PWrite, il comando PRD_RAWWRITE non permette che venga specificato un valore di io_Length pari a −1 (cioè non permette d'inviare una stringa a terminazione nulla), in quanto non è progettato per riconoscere alcuna condizione di EOF. Un comando PRD_RAWWRITE può terminare in anticipo la propria esecuzione soltanto nel caso che si verifichi una condizione d'errore.

PRD_RAWWRITE permette il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non è stato richiesto. L'esito del comando viene restituito nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. PDERR_INTERNALMEMORY indica che non c'era sufficiente memoria per le routine del dispositivo Printer al momento dell'invio del comando PRD_RAWWRITE. IOERR_NOCMD indica che il task ha specificato il parametro io_Command in modo non corretto, e IOERR_BADLENGTH indica che il task ha specificato in modo non corretto io_Length.

Preparazione dell'unione PrinterIO

Il parametro mn_ReplyPort deve contenere l'indirizzo della struttura di tipo MsgPort che rappresenta la reply port del task. I parametri io_Device e io_Unit devono contenere gli indirizzi restituiti dalla funzione OpenDevice. Il task deve impostare io_Command con il comando PRD_RAWWRITE; deve inizializzare io_Flags a IOF_QUICK se intende richiedere il QuickIO; altrimenti deve

inizializzarlo a 0. Inizializzare io_Length con il numero dei caratteri da inviare alla porta seriale o parallela, e infine memorizzare nel parametro io_Data l'indirizzo del buffer che contiene il testo da inviare.

Discussione

Il comando PRD_RAWWRITE permette di trasmettere una serie di codici di controllo alla stampante senza che siano "filtrati" o in qualche modo alterati lungo il tragitto.

12

Il dispositivo Clipboard

Introduzione

Il dispositivo Clipboard gestisce una serie di file clipboard, uno per ogni unità che viene aperta. Un task può scrivere e leggere informazioni (dette anche clip) nei file clipboard, realizzando il cosiddetto "cut & paste". Il dispositivo Clipboard viene programmato tramite diverse funzioni della libreria Exec, quattro comandi standard e tre comandi specifici. È residente su disco.

I dispositivo Clipboard

La Figura 12.1 (a pagina 387) mostra il funzionamento generale del dispositivo Clipboard. Questo dispositivo consente di aprire un numero indefinito di unità, ognuna delle quali è in grado di accodare i comandi in arrivo e di sottoporli all'elaborazione delle routine interne del dispositivo. Ogni volta che viene aperta un'unità, viene creato un nuovo file clipboard. Con il termine "file clipboard" s'intende uno spazio di memoria nel quale il dispositivo Clipboard è in grado di scrivere e leggere insiemi di dati, secondo le richieste dei task presenti nel sistema. Il contenuto di questo particolare tipo di file viene chiamato "clip", e rimane in memoria finché vi rimane il dispositivo.

Il dispositivo è ad accesso condiviso, cioè una stessa unità può essere aperta da diversi task, ognuno dei quali può scrivere e leggere clip. Ovviamente, però, un'unità mantiene solo il clip che ha salvato in memoria più di recente: questo significa che se due task salvano in tempi diversi i loro rispettivi clip nella stessa unità, il clip più recente sovrascrive e quindi elimina quello salvato precedentemente.

I clip privati

I clip possono contenere qualsiasi tipo d'informazione, dalle immagini ai testi, alle musiche. Se un programma crea clip esclusivamente per se stesso, cioè non prevede che i suoi clip possano essere letti anche da altri programmi, può impiegare qualsiasi formato per organizzare i dati al loro interno. Un editor potrebbe per esempio impiegare i file clipboard per muovere blocchi di testo da una zona all'altra del testo che mantiene in memoria, e per compiere questa semplice operazione non ha bisogno né che i clip vengano salvati permanentemente su disco, né che i dati al loro interno siano organizzati secondo un formato standard. Un altro esempio potrebbe essere un editor più evoluto che installa nel sistema un suo task di servizio e invia blocchi di testo a questo task impiegando il dispositivo Clipboard. Anche in questo secondo caso non c'è necessità né che i clip vengano salvati permanentemente su disco né che seguano un formato standard: è sufficiente che l'editor e il task di servizio sappiano che devono utilizzare la stessa unità del dispositivo e lo stesso formato

per organizzare le informazioni all'interno dei clip che creano.

Se si deve impiegare il dispositivo Clipboard per questo tipo di funzioni, è consigliabile servirsi di unità diverse dall'unità zero, alla quale generalmente ricorrono i programmi che prevedono il cut & paste per scambiare informazioni con altri programmi di natura diversa.

Infine, quando un task, dopo aver salvato un clip non standard, provvede a chiudere l'unità, il clip non viene salvato su disco, ma rimane in memoria fino a quando non viene sovrascritto.

I clip pubblici

I due esempi illustrati nel precedente paragrafo mostrano impieghi del dispositivo Clipboard che restringono le comunicazioni ad ambienti non standard e chiusi verso l'esterno. Ma il dispositivo Clipboard è stato ideato soprattutto per offrire a programmi diversi la possibilità di scambiarsi informazioni senza necessariamente "conoscersi".

Perché questo avvenga, occorre che i dati all'interno dei clip vengano organizzati secondo uno standard riconosciuto da tutte le applicazioni che prevedono l'utilizzazione della funzione di cut & paste anche per scambiare dati con altre applicazioni. Lo standard più usato (che viene riconosciuto anche dal dispositivo Clipboard) è l'Interchange File Format, o IFF. Si tratta di uno standard che permette di creare e scambiare file contenenti immagini, testi, musiche e voci sintetizzate. Oltre a questi quattro sotto-formati se ne potrebbero pensare innumerevoli altri, sempre sulla base dello standard IFF, ma non è questa la sede più adatta per approfondire il discorso su tale standard.

Ci limiteremo a dire che tutti i programmi che devono scambiare informazioni con altri programmi simultaneamente presenti in memoria, o anche con programmi mandati in esecuzione in tempi diversi, devono creare i loro clip attenendosi allo standard IFF. Organizzare i dati in questo formato è completamente a carico dei programmi: il dispositivo Clipboard non può fare altro che controllare se un clip da salvare in una sua unità è in formato IFF o meno.

Fino a quando l'unità aperta da un task non viene chiusa, la gestione dei clip pubblici non introduce alcuna differenza rispetto a quella dei clip privati: il clip salvato più di recente viene mantenuto in memoria ed è accessibile a qualunque altro programma che abbia aperto la stessa unità. Le cose cambiano quando il task decide di chiudere l'unità, in quanto se il contenuto del file clipboard è ancora quello che si era salvato (cioè se non è stato sovrascritto da un altro task) e nessun altro task mantiene aperta la stessa unità, il clip, essendo in formato IFF, viene salvato su disco. Il dispositivo Clipboard lo salva nella directory logica CLIPS:, utilizzando come nome il numero dell'unità chiusa dal task. Compiuta questa operazione, ai fini della gestione del clip non è mutato nulla: il clip ha semplicemente cambiato indirizzo, spostandosi dalla memoria al disco. Se successivamente un altro task compie un'operazione di lettura (paste) dalla stessa unità, il dispositivo legge il clip dal file clipboard salvato su disco, sempre che nel frattempo non ne sia stato creato uno nuovo in memoria (che renderebbe obsoleto quello su disco).

Funzionamento del dispositivo Clipboard

Per semplificare la discussione, nella Figura 12.1 viene mostrata soltanto un'unità del dispositivo e un solo task. La coda delle richieste appartenente all'unità viene gestita attraverso la sotto-struttura MsgPort della struttura Unit, inizializzata con la chiamata alla funzione OpenDevice. In questa coda pervengono tutti i comandi inviati dal task e destinati all'unità corrispondente.

Il task deve creare autonomamente due message port utilizzando la funzione CreatePort di supporto alla libreria Exec, e deve allocare per ognuna di esse un bit di segnale. La prima message port dev'essere utilizzata come reply port del task, cioè come message port per ricevere le risposte restituite dal dispositivo in seguito all'elaborazione dei comandi.

La seconda message port, chiamata satisfy message port, dev'essere indicata dal task quando invia il comando CBD_POST (l'indirizzo di questa particolare message port rappresenta a tutti gli effetti il parametro fondamentale della struttura di I/O per questo comando), e viene utilizzata dal dispositivo per inviare ai task i satisfy message: si tratta di un particolare tipo di messaggi che il dispositivo, impiegando la struttura SatisfyMsg, invia a questa particolare message port in particolari condizioni.

Vediamo in quali situazioni il dispositivo inoltra un satisfy message alla satisfy message port del task che gli ha inviato il comando CBD_POST. Le routine interne del dispositivo hanno ricevuto dal task un comando CBD_POST e l'hanno eseguito (e quindi sanno a quale message port inviare il satisfy message). Il comando CBD_POST serve al task per avvisare il dispositivo Clipboard che all'occorrenza è pronto per salvare un clip. In pratica, anziché salvarlo subito, il task si riserva di farlo se gli viene richiesto, evitando così

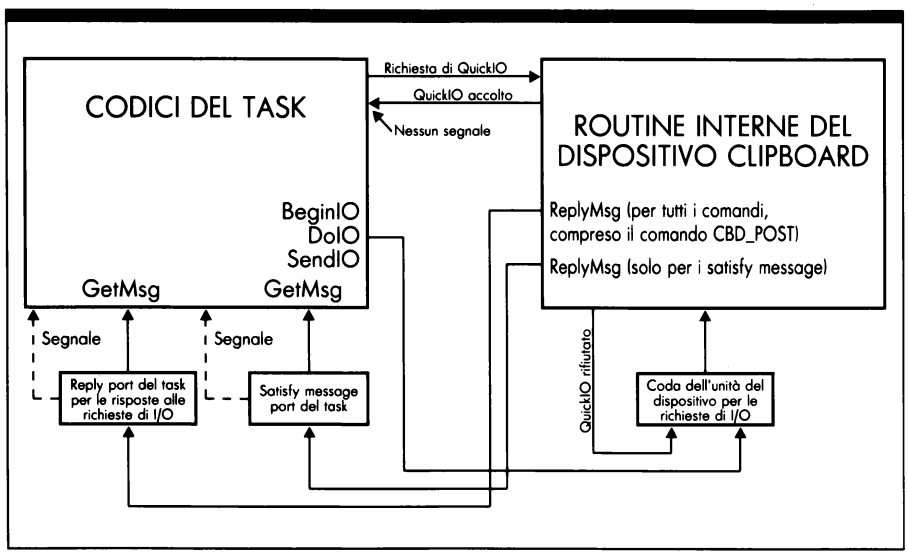


Figura 12.1: *Funzionamento del dispositivo Clipboard*

procedure inutili e dispendiose qualora il clip sia di ampie proporzioni. Eseguita questa operazione, il task deve preoccuparsi di sondare la satisfy message port (come vedremo, prima di ogni rilevazione alla satisfy message port deve però inviare un comando CBD_CURRENTWRITEID, grazie al quale può stabilire se il clip che ha prorogato è ancora attuale).

Se nessun altro task ha salvato clip nella stessa unità, cioè se il clip relativo al comando CBD_POST è ancora il piú recente, il dispositivo invia un satisfy message a quella particolare message port nel momento in cui rileva una richiesta di lettura all'unità (questa richiesta può essere inviata dallo stesso task che ha inoltrato il comando CBD_POST o da un altro). Quando il task riceve nella satisfy message port un satisfy message, deve immediatamente rimuoverlo dalla coda e inoltrare un comando CMD_WRITE per salvare quel clip nel clipboard file dell'unità. Inviando il comando CMD_WRITE, il task che ha messo a disposizione un clip tramite il comando CBD_POST soddisfa la richiesta di un clip inoltrata da un altro task.

Se invece, dopo l'inoltro del comando CBD_POST, un altro task scrive nell'unità un proprio clip, il clip relativo al comando CBD_POST diventa obsoleto, e il primo task non riceverà mai il satisfy message. Il task, per rendersene conto deve assoluatamente evitare di entrare in attesa del satisfy messagge (tramite le funzioni WaitPort o Wait), e prima di chiamare la funzione GetMsg per sondare la satisfy message port deve inoltrare il comando CBD_CURRENTWRITEID. Questo gli consente di confrontare l'identificatore del clip ottenuto dal comando CBD_POST con quello di scrittura conservato all'interno del dispositivo: se sono diversi, significa che è stato scritto un nuovo clip nell'unità, che il clip relativo al comando CBD_POST è obsoleto, e che è inutile continuare a sondare la satisfy message port.

Riassumendo, possiamo dire che un task tramite il comando CBD_POST informa il dispositivo che desidera eseguire un'operazione di cut o copy (scrittura nel file clipboard), ma non procede fino a quando esso stesso, o un altro task, non esegue un'operazione di paste (lettura) sullo stesso file clipboard (cioè sulla stessa unità). In questo senso, il comando CBD_POST agisce come un comando CMD_WRITE a scoppio ritardato, e il comando CBD_CURRENTWRITEID evita al task di entrare in pericolosi loop perpetui.

Gli identificatori dei clip

Uno dei più complicati aspetti della programmazione del dispositivo Clipboard è comprendere il funzionamento degli identificatori dei clip (ordinal clip identifier). *Identificatore del clip* è un altro nome per il parametro io_ClipID della struttura IOClipReq (la struttura impiegata per le richieste di I/O inviate a questo dispositivo; differisce dalla struttura IOStdReq solo per la presenza del parametro io_ClipID). A ogni accesso in lettura o in scrittura all'unità aperta, e quindi al relativo file clipboard, viene sempre associato un identificatore. Questo identificatore è un valore numerico che viene restituito dal dispositivo nel parametro io_ClipID quando un task accede a una delle sue unità, e dev'essere sempre lo stesso per tutti i successivi accessi allo stesso file clipboard (i task possono creare o leggere i file clipboard anche effettuando una

serie di accessi). Nel momento in cui si inizia un accesso all'unità, il dispositivo inizializza il parametro io_ClipID con un nuovo valore. La funzione di questo parametro è quindi identificare univocamente ogni clip che si salva o che si proroga.

Alla stessa unità, e quindi allo stesso file clipboard, non si può accedere simultaneamente in scrittura e in lettura. Qualora sia in corso un accesso in lettura, una richiesta di accesso in scrittura viene posticipata, e viceversa.

Il dispositivo Clipboard mantiene internamente un identificatore del clip per la lettura e un identificatore del clip per la scrittura. Entrambi sono costituiti da numeri interi che inizialmente valgono 1. Le routine interne del dispositivo Clipboard si servono del parametro io_ClipID della struttura IOClipReq per tenere sotto controllo l'ordine con cui sono stati inoltrati i comandi CMD_READ, CMD_WRITE e CBD_POST.

Quando un task intende iniziare un accesso in lettura a un file clipboard (lo stesso procedimento si applica alla scrittura), deve impostare a 0 il parametro io_ClipID e inviare un comando CMD_READ all'unità che ha aperto. Questo comando è in genere il primo di una serie, dal momento che i task generalmente leggono i clip "a gruppi di dati" anziché in un colpo solo. Impartendo il comando, le routine interne del dispositivo Clipboard provvedono ad aggiornare il parametro io_ClipID con un identificatore del clip per questo accesso, che viene definito di lettura. Successive sequenze di comandi CMD_READ allo stesso file clipboard devono essere inviate mantenendo inalterato l'identificatore del clip io_ClipID. In caso di scrittura, s'impiega il comando CMD_WRITE, e l'identificatore del clip che il dispositivo assegna a quest'accesso viene definito di scrittura.

Il task può rilevare l'attuale identificatore del clip di lettura o scrittura (corrispondente al comando CMD_READ o CMD_WRITE inviato più di recente) inviando rispettivamente il comando CBD_CURRENTREADID o il comando CBD_CURRENTWRITEID. Il primo si rende utile ogni qual volta un task desidera eseguire il paste (lettura) di un clip assicurandosi che nel file clipboard relativo all'unità ci sia ancora lo stesso clip che aveva precedentemente salvato durante un'operazione di cut o copy. Il secondo comando, invece, serve al task quando, dopo aver posticipato la scrittura di un clip tramite il comando CBD_POST, deve verificare che non siano stati salvati altri clip nell'unità prima di ricevere la richiesta di quel clip: infatti, se nel frattempo venisse salvato un altro clip nella stessa unità, quella richiesta non giungerebbe mai più. Nell'analisi di questi due comandi riprenderemo in ogni dettaglio il loro funzionamento.

Operazioni sequenziali di lettura e scrittura

La Figura 12.2 (nella pagina successiva) mostra in che modo un task esegue in sequenza una serie di operazioni di lettura e scrittura su un file clipboard. La discussione che segue esamina l'accesso in lettura, ma si applica altrettanto bene all'accesso in scrittura.

Una lettura sequenziale è una serie di comandi CMD_READ consecutivi che un task invia per leggere diversi segmenti di dati dal più recente file

clipboard e per memorizzarli all'interno di un buffer di lettura. Il task potrebbe aver scritto questi dati nel file clipboard in una volta sola oppure con diversi comandi CMD_WRITE; è anche possibile che questi dati siano stati generati e salvati in un clip da un altro task. Nel nostro esempio, i segmenti di dati che compongono il file clipboard sono rispettivamente composti da 8, 12, e 15 byte.

Il task esegue gli accessi in lettura definendo ogni volta i parametri io_Data, io_Length e io_Offset. Per esempio, supponiamo che il task desideri leggere il file clipboard rappresentato nella Figura 12.2 sequenzialmente, partendo dall'inizio del file (la sequenzialità della lettura non è obbligatoria). La procedura che deve seguire è la seguente:

1. Si definisce una struttura IOClipReq per rappresentare il primo comando CMD_READ. Si inizializza il suo parametro io_Data in modo che punti al buffer definito dal task. Quest'operazione definisce le locazioni RAM nelle quali dovranno essere trasferiti i byte contenuti nel file clipboard. Poi si inizializza io_Length a 8, il numero di byte da trasferire nel buffer del task. Si inizializza io_Offset a 0 per indicare che la lettura deve iniziare dal primo byte del file clipboard. Si inizializza a 0 il parametro io_ClipID. Si invia il comando CMD_READ tramite le funzioni BeginIO, DoIO o SendIO. Quando l'esecuzione del comando CMD_READ giunge al termine, le routine interne del dispositivo Clipboard assegnano il valore 8 a io_Offset, che riflette la posizione dell'indice all'interno del file clipboard (individua il byte successivo). Se si desidera proseguire nella lettura sequenziale di altri gruppi di dati, io_Offset indica la posizione iniziale per il successivo comando CMD_READ. Inoltre, dal momento che questo è il primo comando dell'intero accesso, le routine interne del dispositivo Clipboard assegnano un valore numerico al parametro io_ClipID della struttura IOClipReq; questo valore identifica univocamente l'accesso. I successivi comandi CMD_READ devono utilizzare la stessa struttura IOClipReq

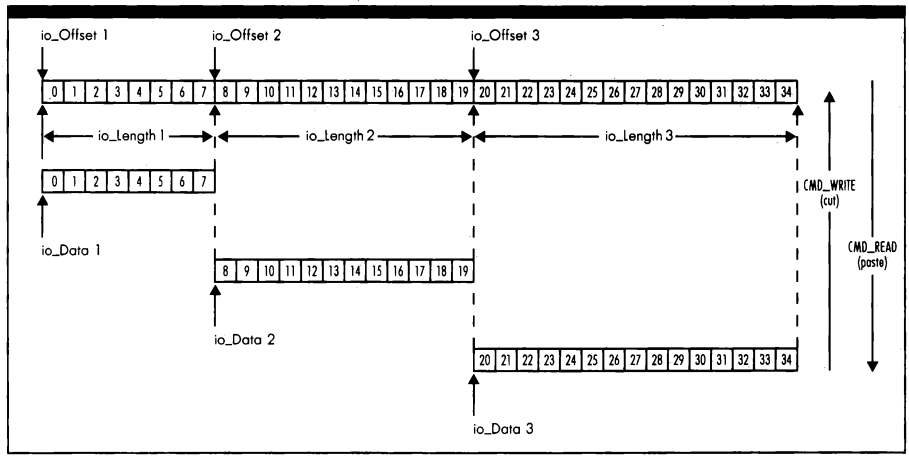


Figura 12.2: *Operazioni di lettura e scrittura sequenziale su un file clipboard*

senza alterare il parametro io_ClipID, in modo che le routine interne siano in grado di riconoscere l'insieme di comandi CMD_READ relativi allo stesso accesso.

2. Si inizializza nuovamente la struttura IOClipReq per rappresentare il secondo comando CMD_READ. Si inizializza io_Data con l'indirizzo del buffer di lettura (può anche essere il buffer già impiegato se il primo segmento di dati è stato già elaborato). Si inizializza io_Length a 12, il numero di byte da trasferire nel buffer del task con il secondo accesso. Non si deve alterare io_Offset se si desidera eseguire una lettura sequenziale del file clipboard, in quanto è già stato automaticamente aggiornato dal dispositivo Clipboard durante l'ultimo accesso. Inoltre, non si deve assolutamente alterare io_ClipID se si vuole continuare con lo stesso tipo di accesso. Si utilizza BeginIO, DoIO o SendIO per inviare il comando. Quando CMD_READ conclude la sua esecuzione, l'indice io_Offset del file clipboard viene aggiornato a 20 (somma delle lunghezze contenute nel parametro io_Length nei due accessi). Nel nostro caso il nuovo valore dell'indice individua il byte 20, il ventunesimo byte del file, che rappresenta la posizione di partenza per il terzo comando CMD_READ.

3. Si inizializza nuovamente la struttura IOClipReq per rappresentare il terzo comando CMD_READ. Si inizializza io_Data con l'indirizzo del buffer di lettura. Si inizializza io_Length a 15, il numero di byte da trasferire nel buffer del task con il terzo accesso in lettura. Ancora una volta, non si deve alterare io_Offset se si desidera eseguire una lettura sequenziale, in quanto è già stato automaticamente aggiornato dal dispositivo Clipboard durante l'ultimo accesso. Inoltre, non si deve assolutamente alterare io_ClipID se si vuole continuare con lo stesso tipo di accesso. Si utilizza BeginIO, DoIO o SendIO per inviare il comando. Quando CMD_READ conclude l'esecuzione, l'indice io_Offset del file clipboard viene aggiornato a 35, valore che individua il byte 35, il trentaseiesimo del file.

4. Per segnalare al dispositivo Clipboard che l'accesso in lettura al file clipboard è terminato, è necessario inviare un ulteriore comando CMD_READ facendo in modo che il parametro indice io_Offset oltrepassi la fine del file clipboard. In questo modo il dispositivo capisce che l'accesso in lettura è terminato.

Ogni comando CMD_READ restituisce nel parametro io_Actual della struttura IOClipReq il numero di byte effettivamente letti. Il task può esaminare questo parametro per verificare se il comando è stato eseguito correttamente. Questa stessa procedura funziona anche per una serie di comandi CMD_WRITE sequenziali. Ovviamente s'inverte la direzione del trasferimento di dati. Il comando CMD_WRITE trasferisce un clip da un buffer del task nel file clipboard dell'unità, e l'indice di questo buffer è rappresentato dal parametro io_Offset. L'unica differenza di rilievo rispetto al comando CMD_READ riguarda la quarta

fase. Infatti, per comunicare al dispositivo che l'accesso in scrittura al file clipboard è terminato, occorre inviare il comando CMD_UPDATE.

comandi del dispositivo Clipboard

Il dispositivo Clipboard possiede tre comandi specifici e quattro comandi standard. Solo il comando CMD_RESET permette il QuickIO, e nessun comando può essere eseguito in modo immediato. Tutti i sette comandi influenzano il parametro io_Error della struttura IOClipReq. Cinque comandi, inoltre, influenzano il parametro io_ClipID della struttura IOClipReq, mentre solo due influenzano i parametri io_Actual e io_Offset.

L'invio dei comandi al dispositivo Clipboard

La Figura 12.3 (nella pagina successiva) mostra in che modo i comandi vengono inviati al dispositivo Clipboard. Le linee con le frecce rappresentano i parametri che si devono inizializzare e quelli restituiti dalle routine interne del dispositivo.

La programmazione del dispositivo Clipboard prevede tre fasi:

1. *Preparazione della struttura*. In questa fase il task possiede il completo controllo. È la fase in cui si inizializzano i parametri nella struttura IOClipReq, prima d'inviare un comando alle routine del dispositivo Clipboard. Questi parametri includono quelli richiesti dalla maggior parte dei dispositivi e alcuni parametri particolari, come io_ClipID e io_Offset. La scelta dei parametri da inizializzare dipende dal comando che s'intende inviare.

2. *Invio del comando e sua elaborazione*. L'unico compito a carico del programmatore in questa fase è quello d'inviare il comando al dispositivo utilizzando le funzioni BeginIO, DoIO o SendIO. Una volta che una di esse è stata mandata in esecuzione, il controllo passa alle routine interne del dispositivo e del sistema.

3. *Elaborazione dei parametri che il dispositivo restituisce*. Il sistema e le routine interne del dispositivo Clipboard possiedono il completo controllo sui valori elaborati in questa fase; i risultati prodotti dall'elaborazione del comando inviato al dispositivo Clipboard vengono restituiti al task che l'aveva inviato. Se la richiesta di I/O non prevede il QuickIO, le risposte si trovano nella coda alla reply port del task. Con il QuickIO, invece, il comando ritorna direttamente al task richiedente.

Diversi comandi del dispositivo Clipboard forniscono parametri in uscita, come viene mostrato nella parte destra della Figura 12.3. Entrambi i comandi CMD_READ e CMD_WRITE restituiscono valori per i parametri io_Error,

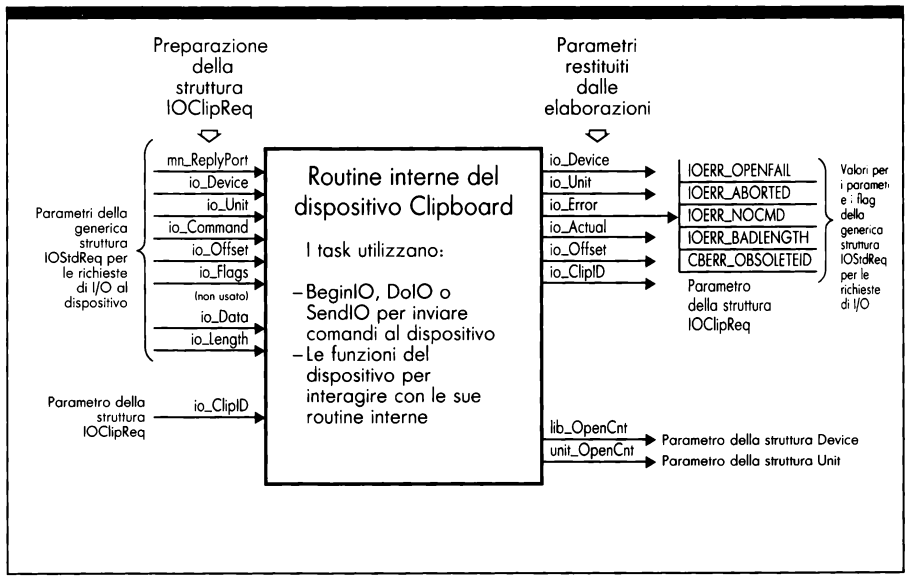


Figura 12.3: *Gestione delle funzioni e dei comandi previsti dal dispositivo Clipboard*

io_Actual, io_Offset e io_ClipID della struttura IOClipReq.

Inoltre, la Figura 12.3 mostra i parametri che ricoprono un ruolo significativo nell'inizializzazione e nell'elaborazione delle funzioni del dispositivo Clipboard. OpenDevice e CloseDevice influenzano, come di consueto, i parametri unit_OpenCnt e lib_OpenCnt appartenenti rispettivamente alle strutture Unit e Device associate con il dispositivo e con l'unità aperta; OpenDevice influenza anche il parametro io_Error.

Le strutture del dispositivo Clipboard

Il dispositivo Clipboard si serve di tre strutture: ClipboardUnitPartial, IOClipReq e SatisfyMsg. La struttura ClipboardUnitPartial viene utilizzata dal dispositivo per mantenere una lista delle sue unità aperte. Ogni chiamata alla funzione OpenDevice per aprire una nuova unità, aggiunge un elemento a questa lista. La struttura IOClipReq viene utilizzata per formulare e inviare tutte le richieste d'esecuzione di comandi alle routine interne del dispositivo Clipboard. La struttura SatisfyMsg viene utilizzata dalle routine interne del dispositivo Clipboard per inviare un satisfy message alla satisfy message port di un task. Quando un task riceve questo messaggio, deve immediatamente inviare un comando CMD_WRITE (precedentemente posticipato da un comando CBD_POST). Queste tre strutture sono mostrate nella Figura 12.4 (a pagina 395).

La struttura ClipboardUnitPartial

La struttura ClipboardUnitPartial è definita come segue:

```
struct ClipboardUnitPartial {
        struct Node cu_Node;
        ULONG cu_UnitNum;
};
```

I suoi parametri hanno il seguente significato:

- cu_Node. Contiene il nome di una sotto-struttura Node utilizzata per inserire la struttura ClipboardUnitPartial all'interno della lista che il dispositivo Clipboard mantiene per tenere traccia delle unità aperte.

- cu_UnitNum. Questo parametro contiene il numero dell'unità del dispositivo Clipboard. Le routine interne del dispositivo assegnano un identificatore numerico a ogni nuova unità aperta.

La struttura IOClipReq

La struttura IOClipReq è definita come segue:

```
struct IOClipReq {
        struct Message io_Message;
        struct Device *io_Device;
        struct Unit *io_Unit;
        UWORD io_Command;
        UBYTE io_Flags;
        BYTE io_Error;
        ULONG io_Actual;
        ULONG io_Length;
        STRPTR io_Data;
        ULONG io_Offset;
        LONG io_ClipID;
};
```

I suoi parametri hanno il seguente significato:

- io_Message. Contiene il nome di una sotto-struttura Message utilizzata per rappresentare la reply port del task. Il parametro mn_ReplyPort della struttura Message punta alla struttura MsgPort che rappresenta la coda alla reply port del task.

- io_Device. Contiene un puntatore alla struttura Device utilizzata per gestire il dispositivo Clipboard.

- io_Unit. Contiene un puntatore alla struttura Unit associata alla particolare unità del dispositivo Clipboard. Il parametro unit_MsgPort della struttura Unit punta alla struttura MsgPort che rappresenta la coda alla request port dell'unità.

- io_Command. Dev'essere aggiornato con il comando che si vuole far eseguire alle routine interne del dispositivo Clipboard.

- io_Flags. Contiene il flag IOF_QUICK.

- io_Error. Contiene il codice d'errore restituito dal dispositivo Clipboard in seguito all'esecuzione di un comando.

- io_Actual. Contiene l'effettivo numero di byte trasferiti dalle routine interne del dispositivo Clipboard durante l'esecuzione di un comando di lettura o scrittura.

- io_Length. Questo parametro rappresenta il numero di byte che un comando CMD_READ o CMD_WRITE deve trasferire tra il file clipboard e il buffer del task.

- io_Data. Con CMD_READ o CMD_WRITE deve contenere l'indirizzo del buffer del task. Nel caso del comando CBD_POST, invece, io_Data deve puntare a una struttura MsgPort che rappresenta la message port destinata a ricevere i satisfy message conseguenti al comando.

- io_Offset. Rappresenta l'indice all'interno del file clipboard o del buffer

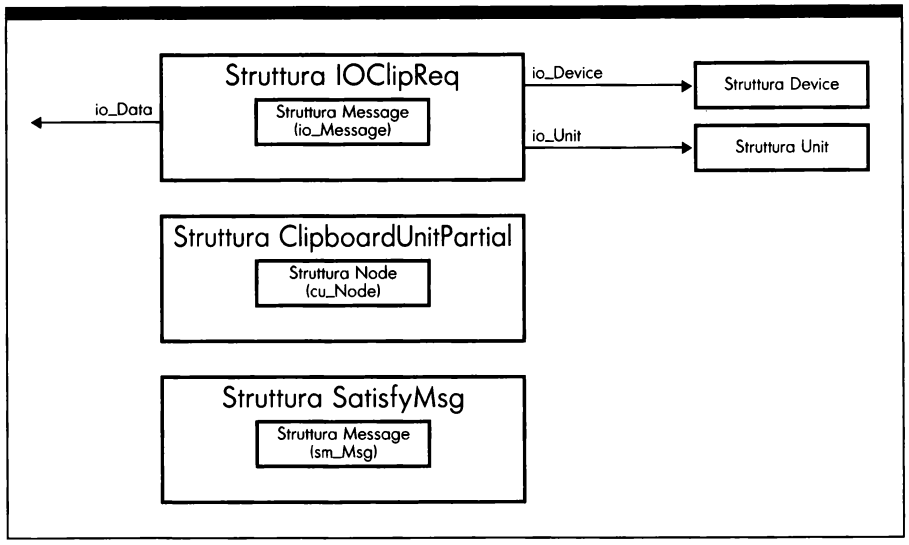


Figura 12.4: *Strutture utilizzate dal dispositivo Clipboard*

del task. In genere il task lo imposta a 0 quando invia il primo comando CMD_READ o CMD_WRITE della sequenza che caratterizza l'accesso; sono poi le routine interne del dispositivo Clipboard che lo aggiornano automaticamente per i successivi comandi.

- io_ClipID. Contiene l'identificatore del clip (ordinal clip identifier) associato a un accesso in corso su un file clipboard. Le routine interne del dispositivo Clipboard specificano questo parametro quando eseguono il primo comando CMD_READ o CMD_WRITE dell'accesso.

La struttura SatisfyMsg

La struttura SatisfyMsg è definita come segue:

```
struct SatisfyMsg {
        struct Message sm_Msg;
        UWORD sm_Unit;
        LONG sm_ClipID;
};
```

I suoi parametri hanno il seguente significato:

- sm_Msg. È il nome della sotto-struttura Message che intesta questo particolare tipo di messaggio. Quando riceve una richiesta di accesso in lettura alla stessa unità, il dispositivo invia questo messaggio alla satisfy message port del task che ha messo a disposizione un clip (tramite il comando CBD_POST). Si noti però che il messaggio viene inviato solo a patto che il clip messo a disposizione dal comando CBD_POST sia ancora il più recente per quell'unità, cioè a patto che nel frattempo in quell'unità non siano stati salvati altri clip.

- sm_Unit. Questo è l'identificatore numerico dell'unità del dispositivo Clipboard alla quale è stato diretto il comando CBD_POST. L'identificatore numerico della principale unità del dispositivo Clipboard è 0. Se il task ha reso disponibili diversi clip aprendo diverse unità, e impiega la stessa message port per ricevere i satisfy message, deve analizzare questo parametro per sapere a quale unità corrisponde il clip richiesto.

- sm_ClipID. Questo è l'identificatore del clip assegnato dalle routine interne del dispositivo Clipboard al comando CBD_POST inviato alla particolare unità. Il valore assegnato a questo parametro mantiene il collegamento fra l'originale comando CBD_POST e il satisfy message ricevuto dal task.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOClipReq)
A1

Scopo della funzione

CloseDevice chiude per il task l'accesso a un'unità del dispositivo Clipboard. I parametri io_Device e io_Unit della struttura IOClipReq vengono impostati a −1, a indicare che la struttura IOClipReq non può più essere utilizzata fino a quando questi parametri non verranno nuovamente inizializzati con una chiamata a OpenDevice. Inoltre, CloseDevice decrementa il parametro unit_OpenCnt della struttura Unit e il parametro lib_OpenCnt della struttura Device. Quando i parametri unit_OpenCnt corrispondenti a tutte le unità aperte del dispositivo valgono zero, e conseguentemente il parametro lib_OpenCnt vale anch'esso 0, il dispositivo viene eliminato dalla memoria (purché sia presente una precedente richiesta di eliminazione).

Quando esegue CloseDevice, un task non può utilizzare il dispositivo Clipboard finché non chiama OpenDevice, tuttavia vengono salvati i parametri interni del dispositivo Clipboard che possono essere utilizzati per la successiva chiamata a OpenDevice.

Argomenti della funzione

iOClipReq — Questo argomento dev'essere l'indirizzo della struttura IOClipReq con la quale è stata aperta l'unità che ora si desidera chiudere.

Discussione

CloseDevice chiude per il task l'accesso a un'unità del dispositivo Clipboard. Se al momento della chiusura dell'unità per il task non risultano

aperte altre unità dello stesso dispositivo, quest'ultimo viene chiuso definitivamente.

Un task dovrebbe sempre verificare che l'unità abbia restituito tutte le risposte alle sue richieste di I/O, prima di chiamare CloseDevice. Per effettuare questo controllo può utilizzare le funzioni GetMsg, Remove, CheckIO e WaitIO.

Se il task chiude un'unità nel cui file clipboard ha precedentemente inviato un clip, se tale clip è ancora il più recente per quell'unità ed è organizzato secondo il formato IFF, e se l'unità non è condivisa da nessun altro task, CloseDevice fa in modo che il dispositivo salvi il file clipboard su disco, all'interno della directory logica CLIPS:; il file che viene creato su disco ha come nome il numero dell'unità. In questo modo, il file clipboard sopravvive alla chiusura totale dell'unità, e anche del dispositivo. Se successivamente un task apre la stessa unità e compie un accesso in lettura, il dispositivo accede a quel file su disco (purché non siano stati effettuati altri accessi in scrittura e il file clipboard si possa ancora considerare il più recente per l'unità indirizzata). Infine, si noti che il dispositivo salva su disco il file clipboard dell'unità anche se l'accesso in scrittura per creare tale file non è stato correttamente concluso con il comando CMD_UPDATE. Si ricordi però che se il file clipboard non è in formato IFF e l'unità non risulta aperta anche da altri task, chiudendola con la funzione CloseDevice il file clipboard viene irrimediabilmente perso.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("clipboard.device", unitNumber, iOClipReq, ØL)
DØ                 AØ                   DØ          A1         D1
```

Scopo della funzione

Questa funzione apre per il task l'accesso a una specifica unità del dipositivo Clipboard. Provvede a incrementare il parametro lib_OpenCnt della struttura Device, e il parametro unit_OpenCnt della struttura Unit per indicare che un task in più sta utilizzando l'unità del dispositivo. Le unità del dispositivo Clipboard vengono aperte sempre nel modo di accesso condiviso. In questo modo, più task possono scambiarsi dati attraverso gli stessi file clipboard.

OpenDevice richiede una reply port del task inizializzata in modo appropriato e un bit di segnale ivi allocato se il task desidera essere avvertito ogni volta che giunge un messaggio alla message port. I risultati prodotti dall'esecuzione della funzione vengono restituiti nei parametri che seguono.

- io_Device. Contiene l'indirizzo della struttura Device che il sistema impiega per interagire con la libreria di routine del dispositivo.
- io_Unit. Contiene l'indirizzo della struttura Unit che individua univocamente l'unità prescelta dal task.
- io_Error. Un valore di questo parametro pari a 0 indica che la richiesta di apertura del dispositivo ha avuto successo. IOERR_OPENFAIL indica che non è stato possibile aprire il dispositivo; di solito questo tipo di errore si verifica quando non c'è memoria sufficiente.

Argomenti della funzione

"clipboard.device" Il task deve indicare in questo argomento la stringa contenente il nome del dispositivo.

unitNumber Indica il numero dell'unità del dispositivo Clipboard che si intende aprire; è necessario utilizzare sempre l'unità 0 se si vogliono creare e leggere clip pubblici (nel file INCLUDE del dispositivo viene definita la costante PRIMARY_CLIP per indicare negli argomenti della funzione l'unità 0). Se si vogliono creare e leggere clip privati, si può impiegare un qualsiasi numero diverso da 0.

iOClipReq Deve contenere l'indirizzo della struttura di tipo IOClipReq creata dal task per aprire il dispositivo.

ØL Indica che la funzione ignora l'argomento flag.

Preparazione della struttura IOStdReq

Per aprire il dispositivo Clipboard occorre inizializzare il parametro mn_ReplyPort della struttura di I/O con l'indirizzo della struttura MsgPort che rappresenta la reply port del task. Il task può allocare una message port tramite la funzione CreatePort di supporto alla libreria Exec, e indicarne l'indirizzo come argomento della funzione CreateExtIO, sempre di supporto alla libreria Exec. Chiamando quest'ultima funzione il task alloca la struttura di I/O necessaria per interagire con il dispositivo e memorizza automaticamente l'indirizzo della reply port nel parametro mn_ReplyPort.

Discussione

OpenDevice apre per un task l'accesso a un'unità del dispositivo Clipboard, e inizializza i parametri della struttura IOClipReq necessari per inviare comandi al dispositivo. Ogni parametro può essere inizializzato dalle routine del dispositivo a un valore di default, oppure può mantenere il valore che aveva prima della chiamata. Se il task desidera utilizzare altri valori per questi parametri, deve provvedere a inizializzarli non appena termina l'esecuzione di OpenDevice. Quando un task non deve più accedere a Clipboard dovrebbe provvedere alla sua chiusura tramite CloseDevice.

COMANDI STANDARD DEL DISPOSITIVO

CMD_READ

Scopo del comando

CMD_READ provoca il trasferimento di un flusso di caratteri dal file clipboard che corrisponde all'unità prescelta al buffer definito dal task. Il flusso viene denominato "clip", e l'operazione di lettura viene definita "paste". Il numero di caratteri che si vogliono leggere dev'essere specificato nel parametro io_Length della struttura IOClipReq, il byte all'interno del file clipboard dal quale si vuole iniziare la lettura dev'essere indicato nel parametro io_Offset che funge da indice (il primo byte del file corrisponde sempre all'indice zero), e l'indirizzo del buffer del task dev'essere indicato dal parametro io_Data. CMD_READ accede in lettura al file clipboard dell'unità, che può trovarsi in memoria o su disco (in questo caso è necessariamente organizzato secondo il formato IFF).

Dato che CMD_READ non prevede il QuickIO, viene sempre restituito alla reply port del task. I risultati prodotti dall'esecuzione del comando vengono restituiti nei parametri io_Actual, io_ClipID e io_Error. Il numero di caratteri effettivamente letti dal file clipboard tramite il comando CMD_READ viene restituito nel parametro io_Actual. L'identificatore del clip viene specificato dal dispositivo nel parametro io_ClipID qualora il comando CMD_READ sia il primo accesso in lettura da parte del task (in questo caso, io_ClipID doveva essere a zero prima della chiamata); questo valore dev'essere poi utilizzato con tutti i successivi comandi CMD_READ, fino a quando l'accesso non è concluso. Se invece il comando CMD_READ non è il primo di una serie, le routine interne del dispositivo non alterano il parametro io_ClipID. Il valore 0 nel parametro

io_Error indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command non è stato specificato correttamente; IOERR_BADLENGTH indica che non è stato specificato correttamente il parametro io_Length.

Preparazione della struttura IOClipReq

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si deve inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo Clipboard e l'unità; questi parametri devono essere copiati dalla struttura IOClipReq inizializzata con la prima chiamata alla funzione OpenDevice. Il parametro io_Command va inizializzato a CMD_READ e io_Flags a 0. Si devono inizializzare inoltre i seguenti parametri:

- io_Length. Deve contenere il numero di caratteri di cui si richiede la lettura dal file clipboard e il trasferimento nel buffer del task.

- io_Data. Deve contenere un puntatore al primo byte del buffer di lettura del task. Se viene inizializzato a 0, le routine interne del dispositivo Clipboard incrementano il parametro io_Offset con il valore espresso da io_Length, come se il comando CMD_READ avesse davvero letto il numero di byte specificati da io_Length. Questo è il modo in cui un task può "leggere" sequenzialmente il file clipboard per raggiungere un particolare segmento di dati. Se il task imposta il parametro io_Length con un valore tale da oltrepassare all'indice io_Offset la fine del file clipboard, l'invio del comando CMD_READ segnala al dispositivo che l'accesso in lettura è terminato.

- io_Offset. Se si desidera leggere un particolare segmento di dati all'interno del file clipboard, si deve inizializzare questo parametro con il numero del primo byte del segmento (che non dev'essere necessariamente il primo byte del file clipboard). Se invece si desidera leggere il file clipboard dall'inizio, il parametro deve contenere il valore 0 per indicare il primo byte del file. Alla fine di ogni lettura (cioè di ogni comando CMD_READ), il dispositivo aggiorna il parametro io_Offset in modo che punti al primo byte non ancora letto. In questo modo, se si desidera compiere un accesso in lettura ai dati che seguono il segmento già trasferito, il parametro io_Offset (l'indice all'interno del file) è già pronto. Quando io_Offset punta a un byte che si trova oltre la fine del file clipboard, inviando il comando CMD_READ si segnala alle routine interne del dispositivo che l'accesso in lettura deve considerarsi concluso.

- io_ClipID. Si deve inizializzarlo a 0 se il comando CMD_READ è il primo di una serie di accessi in lettura. Le routine interne del dispositivo

Clipboard provvedono ad aggiornare questo parametro perché individui univocamente l'accesso finché il task non ne segnala la conclusione (prima che un accesso in lettura termini possono essere inviati parecchi comandi CMD_READ).

Discussione

Come si può notare nella Figura 12.2 (a pagina 390), il file clipboard può essere letto in modo sequenziale. Se il parametro io_Offset indicato impartendo il comando contiene il valore 0, la lettura inizia dal primo byte. Il task deve inizializzare il parametro io_ClipID della struttura IOClipReq a 0 se il comando CMD_READ che invia è il primo di un accesso in lettura; sono le routine interne del dispositivo Clipboard che provvedono ad assegnare un opportuno valore al parametro in modo da identificare univocamente l'accesso. Questo valore, che abbiamo chiamato l'"identificatore" del clip, dev'essere poi indicato con ogni comando CMD_READ dello stesso accesso in lettura.

Se un task inizializza a 0 il parametro io_Data della struttura IOClipReq, le routine interne del dispositivo Clipboard incrementano il parametro io_Offset della struttura IOClipReq con il valore del parametro io_Length. In questo modo un task può muovere l'indice lungo il file, ed eventualmente oltre l'ultimo byte del file. In questo caso, inviare il comando CMD_READ serve per segnalare la conclusione dell'accesso in lettura.

Quando un task sta compiendo un accesso in lettura, qualsiasi tentativo di accesso in scrittura alla stessa unità verrà rimandato fino a quando non viene indicato al dispositivo che l'accesso in lettura è finito. Infine, occorre notare che se il dispositivo riceve un comando CMD_READ e per la stessa unità rileva che è in corso un processo di scrittura (CMD_WRITE), non restituisce la richiesta relativa a CMD_READ fino a quando quell'accesso non si conclude (con l'arrivo del comando CMD_UPDATE). Se invece il dispositivo dovesse rilevare che non è presente nessun file clipboard nell'unità indirizzata, restituisce la richiesta con il valore −1 nel parametro io_ClipID. Si badi che in questo caso il parametro io_Error non contiene alcun codice d'errore.

CMD_RESET

Scopo del comando

CMD_RESET riporta l'unità specificata alle condizioni iniziali. Tutti i parametri delle routine interne del dispositivo Clipboard vengono riportati ai rispettivi valori di default. Soltanto i parametri io_Device e io_Unit inizializzati dalla chiamata alla funzione OpenDevice rimangono inalterati.

CMD_RESET permette il QuickIO e viene restituito alla reply port del task soltanto se il flag IOF_QUICK non viene impostato. I risultati prodotti dell'esecuzione del comando si trovano nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOClipReq

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura IOClipReq inizializzata con la prima chiamata alla funzione OpenDevice. Si devono infine inizializzare io_Command a CMD_RESET e io_Flags a 0, oppure a IOF_QUICK se si desidera il QuickIO.

Discussione

Il comando CMD_RESET compie una semplice operazione: riporta i parametri delle routine interne del dispositivo Clipboard ai rispettivi valori di default.

CMD_UPDATE

Scopo del comando

CMD_UPDATE informa un'unità del dispositivo Clipboard che l'attuale accesso in scrittura al suo file clipboard, cioè la sequenza di comandi CMD_WRITE, è terminato: l'unità può iniziare un eventuale accesso in lettura precedentemente sospeso. CMD_UPDATE non ha effetto se nella coda alla request port dell'unità si trova ancora un comando CMD_WRITE, oppure se le risposte ai comandi CMD_WRITE non sono state ancora restituite tutte alla reply port del task.

Dato che CMD_UPDATE non permette il QuickIO, viene sempre restituito alla reply port del task. Un valore 0 del parametro io_Error indica che il comando è stato eseguito; IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOClipReq

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura IOClipReq inizializzata con la prima chiamata alla funzione OpenDevice. Si devono inizializzare infine io_Command a CMD_UPDATE e io_Flags a 0.

Discussione

Un task invia CMD_UPDATE dopo uno o più comandi CMD_WRITE e dopo aver verificato che tutte le richieste sono tornate alla sua reply port. Per farlo, si possono utilizzare le funzioni GetMsg, Remove, CheckIO e WaitIO (purché non vi siano state chiamate sincrone). Inviando CMD_UPDATE si segnala al dispositivo che deve considerare concluso l'accesso in scrittura.

CMD_WRITE

Scopo del comando

CMD_WRITE provoca il trasferimento di un insieme di dati dal buffer del task nel file clipboard dell'unità indirizzata. Il flusso di dati viene chiamato clip, e l'operazione di scrittura viene definita "cut" o "copy". Il numero dei caratteri da trasferire viene specificato nel parametro io_Length della struttura IOClipReq, mentre il byte del file clipboard dal quale deve iniziare il trasferimento dev'essere indicato nel parametro indice io_Offset. L'indirizzo del buffer del task dev'essere specificato dal parametro io_Data.

Quando si inizia un nuovo accesso in scrittura occorre azzerare il parametro io_ClipID, nel quale il dispositivo restituisce un "identificatore" che individua univocamente l'accesso. I successivi comandi CMD_WRITE devono indicare questo parametro se desiderano proseguire l'accesso. Tuttavia, se un task invia un comando CMD_WRITE in risposta a un satisfy message giunto alla satisfy message port (l'arrivo del messaggio può essere rilevato tramite un segnale che la reply port invia al task), il parametro io_ClipID della struttura di I/O che definisce il comando dev'essere quello indicato nel parametro sm_ClipID della struttura SatisfyMsg che definisce il satisfy message. Questo messaggio viene inviato al task dal dispositivo quando quest'ultimo riceve una richiesta di dati (paste) e il task aveva messo a disposizione un clip, tramite il comando

CBD_POST, che risulta ancora il più recente per l'unità indirizzata.

CMD_WRITE non permette il QuickIO e quindi viene sempre restituito alla reply port del task. I risultati prodotti dall'esecuzione del comando vengono restituiti nei seguenti parametri:

- io_Actual. Indica il numero di caratteri effettivamente scritti nel file clipboard.

- io_ClipID. Viene inizializzato automaticamente dalle routine interne del dispositivo Clipboard durante l'esecuzione del primo comando CMD_WRITE. Tutti i successivi comandi CMD_WRITE devono mantenerlo inalterato.

- io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto mentre IOERR_BADLENGTH indica che è stato specificato in modo non corretto il parametro io_Length.

Preparazione della struttura IOClipReq

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura IOClipReq inizializzata con la prima chiamata alla funzione OpenDevice. Si devono inizializzare infine io_Command a CMD_WRITE e io_Flags a 0, nonché i seguenti parametri:

- io_Length. Deve indicare il numero di caratteri da trasferire nel file clipboard corrispondente all'unità aperta.

- io_Data. Deve individuare l'indirizzo del buffer di scrittura del task.

- io_Offset. Rappresenta l'indice con il quale il dispositivo scorre il file clipboard. Dev'essere inizializzato a 0 perché il trasferimento di dati parta dall'inizio del file clipboard. Per le operazioni di scrittura successive, le routine interne del dispositivo Clipboard provvedono ad aggiornare automaticamente questo parametro con la posizione del byte successivo all'ultimo trasferito. Se io_Offset viene specificato con un valore che oltrepassa le dimensioni del clip, il clip viene completato da una serie di zeri.

- io_ClipID. Dev'essere inizializzato a 0 se il comando CMD_WRITE è il primo di un accesso in scrittura. Le routine interne del dispositivo Clipboard assegnano a questo parametro un valore che individua univocamente l'accesso in scrittura. Il task può leggerlo accedendo alla

struttura IOClipReq restituita dall'esecuzione del comando. Una volta assegnato, il parametro io_ClipID dev'essere impiegato in tutti i comandi CMD_WRITE che seguiranno.

Nel caso invece che il comando CMD_WRITE venga inviato dal task in seguito all'arrivo nella sua satisfy message port di un satisfy message, il valore da inserire nel parametro io_ClipID è quello presente nel parametro sm_ClipID della struttura SatisfyMsg che definisce il satisfy message.

Discussione

Il comando CMD_WRITE permette di scrivere nel file clipboard dell'unità un insieme di dati (clip). Può trattarsi di un accesso singolo o multiplo.

Il task deve azzerare il parametro io_ClipID della struttura IOClipReq prima d'inviare il primo comando CMD_WRITE; le routine interne del dispositivo Clipboard provvedono poi ad assegnare un identificatore del clip a questo parametro, in modo da caratterizzare l'accesso univocamente. L'identificatore del clip deve rimanere inalterato in tutti i comandi CMD_WRITE che vengono inviati successivamente per completare la scrittura del clip.

I dati che costituiscono il clip possono essere organizzati senza seguire uno standard particolare, ed essere quindi destinati a un uso interno, oppure possono essere organizzati secondo lo standard IFF, che è stato creato proprio per rendere possibile lo scambio d'informazioni fra applicazioni che "non si conoscono".

Se il clip non segue il formato IFF, viene perduto al momento della chiusura definitiva dell'unità; infatti il dispositivo Clipboard salva i file clipboard (nella directory logica CLIPS:) solo quando i dati al loro interno sono organizzati secondo il formato IFF.

Non descriviamo il formato IFF, ma mostriamo un esempio di clip organizzato secondo questo standard che può essere letto (paste) dal tool Notepad. Supponendo di voler salvare in un file clipboard il testo "Clip", occorre inizializzare il parametro io_Data della richiesta di I/O come segue:

```
iOClipReq->io_Data = (STRPTR)
    "FORM\x00\x00\x00\x10FTXTCHRS\x00\x00\x00\x04Clip";
```

Nella stringa che costituisce il clip, "FORM" identifica il formato IFF di organizzazione dei dati, la successiva long word indica il numero di caratteri che seguono, "FTXTCHRS" indica che la sezione di dati costituisce un testo formattato (cioè in grado di contenere caratteri particolari di cambio fonte, corpo, impaginazione...), la long word successiva indica il numero di caratteri del testo, e infine il testo vero e proprio. Questo è solo un semplice esempio di clip conforme allo standard IFF, che permette di organizzare anche strutture molto complesse. Ai fini della nostra trattazione è importante soprattutto sottolineare che un clip in formato IFF è accessibile ad applicazioni diverse e viene salvato su disco, cioè non viene perso, se si chiude l'unità.

COMANDI SPECIFICI DEL DISPOSITIVO

CBD_CURRENTREADID

Scopo del comando

Il comando CBD_CURRENTREADID restituisce nel parametro io_ClipID della richiesta di I/O l'identificatore di lettura interno mantenuto dal dispositivo per l'unità indirizzata. L'utilità di questo comando si dimostra tutte le volte che un task utilizza il dispositivo Clipboard non solo per clip pubblici, ma anche per clip privati, cioè per effettuare il paste di clip che aveva esso stesso salvato. Leggendo l'identificatore interno di lettura prima di effettuare un'operazione di lettura, il task ha infatti l'opportunità di confrontarlo con quello che aveva ottenuto salvando un clip (o prorogandolo con il comando CBD_POST) e quindi stabilire se quel clip esiste ancora (nel caso che sia stato salvato) o se è ancora il più recente (nel caso che sia stato prorogato).

CBD_CURRENTREADID non permette il QuickIO e quindi viene sempre restituito alla reply port del task. Un valore 0 di io_Error indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOClipReq

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura IOClipReq inizializzata con la prima chiamata alla funzione OpenDevice. Si devono infine inizializzare io_Flags, io_Length, io_Data, io_Offset e io_ClipID a 0, e io_Command a CBD_CURRENTREADID.

Discussione

Il comando CBD_CURRENTREADID permette a un task di sapere qual è l'identificatore di lettura mantenuto dal dispositivo per l'unità indirizzata. Un esempio rende particolarmente semplice comprendere l'utilità di questo comando.

Supponiamo che un'applicazione predisposta per effettuare operazioni di

cut & paste debba eseguire un'operazione di cut (scrittura) e che il file clipboard che genera sia pubblico, cioè accessibile anche ad altre applicazioni. Proprio perché dotata anche dell'opzione paste, può capitare che l'utente richieda il paste del clip appena salvato, magari per spostare un blocco di testo in un altro punto di un file. Per l'applicazione, prima d'inviare il comando CMD_READ, nasce l'esigenza di accertare che il clip mantenuto dall'unità sia quello che aveva salvato, e non un altro magari salvato nella stessa unità da un'altra applicazione. Per compiere questo controllo, l'applicazione deve innanzitutto mantenere in una variabile l'identificatore del clip che aveva ottenuto dal dispositivo inviando il comando CMD_WRITE. Successivamente, prima d'inviare il comando CMD_READ inoltra CBD_CURRENTREADID e confronta l'identificatore interno di lettura con quello che ha mantenuto. Se sono uguali significa che inoltrando il comando CMD_READ legge lo stesso clip che aveva salvato. Se invece sono diversi, significa che quel clip è stato sovrascritto da uno più recente, ed evidentemente non è il caso di leggerlo.

CBD_CURRENTWRITEID

Scopo del comando

Il comando CBD_CURRENTWRITEID restituisce nel parametro io_ClipID della richiesta di I/O l'identificatore di scrittura interno mantenuto dal dispositivo per l'unità indirizzata. Si tratta dell'identificatore relativo al più recente clip salvato tramite CMD_WRITE o prorogato tramite CBD_POST. Questo identificatore può essere confrontato con quello di un precedente comando CBD_POST: se quest'ultimo risulta minore significa che il clip mantenuto privatamente nello spazio di memoria del task e messo a disposizione per eventuali operazioni di paste non è più attuale in quanto un altro task ha successivamente salvato in quell'unità un altro clip.

CBD_CURRENTWRITEID non permette il QuickIO e quindi viene sempre restituito alla reply port del task. Il valore 0 in io_Error indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOClipReq

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura IOClipReq inizializzata con la prima chiamata alla

funzione OpenDevice. Si devono inizializzare infine io_Flags, io_Length, io_Data, io_Offset e io_ClipID a 0, e io_Command a CBD_CURRENTWRITEID.

Discussione

Il comando CBD_CURRENTWRITEID permette a un task di rilevare l'identificatore di scrittura mantenuto internamente dal dispositivo per l'unità indirizzata. Un esempio rende particolarmente semplice comprendere l'utilità di questo comando.

Supponiamo che un task desideri mettere a disposizione del sistema un clip, salvandolo nel file clipboard dell'unità solo se qualcuno lo richiede. Per far questo proroga il clip tramite il comando CBD_POST. Successivamente, il task deve predisporsi per rilevare l'arrivo di un satisfy message alla sua satisfy message port (l'arrivo di questo messaggio, infatti, indica che qualcuno ha cercato di accedere in lettura alla stessa unità e che quindi occorre impartire un comando CMD_WRITE per salvare il clip). Per farlo può entrare in attesa chiamando la funzione WaitPort e indicando come argomento l'indirizzo della satisfy message port, ma questa operazione presenta ovviamente due grandi rischi. 1) Nessuno potrebbe effettuare un accesso in lettura a quell'unità. 2) Qualcuno potrebbe aver salvato un nuovo clip nell'unità, rendendo quindi obsoleto quallo mantenuto dal task e prorogato con il comando CBD_POST. In entrambi i casi il task non riottiene più il controllo in quanto il dispositivo non invia il satisfy message. Inoltre, mentre nel primo caso non è comunque detto che non possa prima poi arrivare, nel secondo è assolutamente sicuro che non arriverà.

La situazione, comunque, non cambierebbe se il task entrasse in un loop chiamando ciclicamente la funzione GetMsg per sondare la satisfy message port, in quanto non potrebbe mai scoprire che il suo clip prorogato è diventato obsoleto.

È evidente quindi la necessità di uno strumento che consenta al task di rilevare se il clip prorogato è ancora il più recente. Se nel ciclo che periodicamente sonda la satisfy message port inoltra anche il comando CBD_CURRENTWRITEID e confronta l'identificatore di scrittura interno per quell'unità con quello che aveva ottenuto inviando il comando CBD_POST, può a un certo momento rilevare che il clip prorogato è ormai superato, e che quindi non giungerà mai nessun satisfy message. Se questo accade, è del tutto inutile continuare a sondare la satisfy message port, ed è opportuno inviare nuovamente il comando CBD_POST.

CBD_POST

Scopo del comando

Un task tramite il comando CBD_POST informa il dispositivo che è disponibile un clip di dati per eventuali operazioni di paste, cioè che è pronto a trasferirlo nel file clipboard tramite il comando CMD_WRITE (operazione di cut), ma che non procederà fino a quando un altro task, o lo stesso dispositivo, non cercherà di eseguire un'operazione di paste (lettura) dalla stessa unità. In questo senso, il comando CBD_POST agisce come un comando CMD_WRITE a scoppio ritardato. Il vantaggio è che in questo modo il trasferimento del clip nel file clipboard avviene soltanto se è apertamente richiesto, evitando così operazioni che possono rivelarsi inutili e dispendiose, soprattutto quando il clip è di ampie dimensioni.

Dato che CBD_POST non permette il QuickIO, viene sempre restituito alla reply port del task. Un valore 0 nel parametro io_Error indica che il comando è stato eseguito; IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura IOClipReq

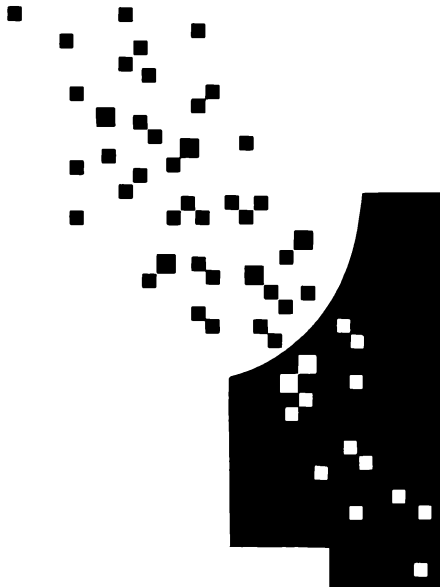
Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura IOClipReq inizializzata con la prima chiamata alla funzione OpenDevice. E ancora, inizializzare io_Command a CBD_POST e io_Flags, io_Length, io_Offset e io_ClipID a 0. Si deve inizializzare infine io_Data in modo che punti a una struttura di tipo MsgPort che rappresenta la satisfy message port per i satisfy message; per creare questa porta si utilizza la funzione CreatePort. Questa message port può anche appartenere a un task diverso da quello che invia il comando CBD_POST.

Discussione

CBD_POST permette a un task di rimandare la scrittura di un clip fino a quando lo stesso task (o un altro) non invia all'unità una richiesta di lettura (paste). In questo modo, un task può inviare un clip al file clipboard gestito da un altro task, ossia un task può impartire un comando CMD_WRITE per un altro task. Per quanto riguarda l'organizzazione dei dati all'interno del clip,

rimandiamo il lettore alla spiegazione del comando CMD_WRITE.

Quando le routine interne del dispositivo Clipboard rilevano che devono soddisfare la richiesta di un comando CMD_READ (un task ha chiesto di accedere in lettura al file clipboard), inviano al task che possiede la satisfy message port un satisfy message. Il task, ricevuto il messaggio, deve salvare il clip tramite un comando CMD_WRITE, per il quale specifica l'identificatore indicato dal parametro sm_ClipID della struttura SatisfyMsg ricevuta. In seguito questa struttura dev'essere rimossa dalla coda alla satisfy message port del task. Se arriva il satisfy message, il task ha la certezza che il clip prorogato è ancora il più recente, e non deve quindi operare altre verifiche. Ma il satisfy message può anche non arrivare mai se nel frattempo un altro task ha salvato un nuovo clip nella stessa unità, rendendo obsoleto quello relativo al comando CBD_POST, oppure se nessun task compie un accesso in lettura all'unità. Per questa ragione, è opportuno che il task non entri in attesa del satisfy message tramite una funzione come WaitPort o Wait, perché altrimenti potrebbe non riprendere più il controllo. Inoltre il task deve inserire nel ciclo che sonda la satisfy message port anche i codici per inviare il comando CBD_CURRENTWRITEID al fine di rilevare se il clip prorogato è ancora il piu recente, come viene ampiamente spiegato nella descrizione di questo comando.



13

Il dispositivo Timer

Introduzione

Il dispositivo Timer fornisce un meccanismo di temporizzazione per i task. Risiede su ROM, e nel caso dell'Amiga 1000 viene caricato dal disco del Kickstart nella ROM WCS durante l'attivazione del sistema. Il dispositivo Timer viene aperto automaticamente dall'AmigaDOS e dai dispositivi Parallel, Serial, Console e Input.

La sua principale funzione è quella d'inviare un segnale a un task quando è trascorso un periodo di tempo stabilito. Tuttavia, dato che l'Amiga è un sistema multitasking, il dispositivo Timer non sempre può garantire che sia trascorso l'*esatto* intervallo di tempo richiesto; il tempo effettivamente trascorso, comunque, non sarà mai minore di quello richiesto.

Per programmare il dispositivo Timer si utilizzano le funzioni OpenDevice, CloseDevice e tre funzioni specifiche: AddTime, CmpTime e SubTime, che permettono di eseguire operazioni aritmetiche con il tempo di sistema (il numero di secondi trascorsi dall'accensione della macchina o dall'ultimo aggiornamento), al fine di compensare gli errori nei conteggi dovuti ai livelli di occupazione del sistema, e rendere più preciso il dispositivo Timer nell'eseguire le richieste di temporizzazione. Inoltre, tre comandi specifici – TR_ADDREQUEST, TR_GETSYSTIME e TR_SETSYSTIME – permettono a un task d'impostare intervalli di tempo, di avere informazioni sul tempo di sistema e di cambiarlo.

Il tempo di sistema è espresso in secondi e microsecondi. Viene incrementato a ogni interrupt di vertical-blanking (l'interrupt generato ogni volta che lo schermo entra in blank, cioè ogni volta che il pennello elettronico che lo scandisce completa un quadro e si appresta a iniziarne uno nuovo). Questo interrupt si verifica ogni sessantesimo di secondo nel sistema americano, e ogni cinquantesimo di secondo nel sistema europeo.

Funzionamento del dispositivo Timer

La Figura 13.1 (nella pagina successiva) illustra il funzionamento generale del dispositivo Timer. Possiede due unità; l'unità 0, UNIT_MICROHZ, è dedicata a temporizzazioni dell'ordine del microsecondo, e perde di accuratezza nella misura d'intervalli di tempo molto più lunghi. L'unità 1, UNIT_VBLANK, è invece dedicata a temporizzazioni dell'ordine del cinquantesimo di secondo, ma presenta un'elevata accuratezza anche per intervalli più lunghi. Entrambe le unità possono essere aperte soltanto nel modo di accesso condiviso.

Le richieste vengono inviate alle routine interne del dispositivo Timer utilizzando le funzioni DoIO o SendIO; la funzione BeginIO non viene utilizzata con questo dispositivo. DoIO serve per le richieste sincrone, nelle quali il task rimane in attesa che il dispositivo Timer termini l'elaborazione e restituisca la relativa risposta.

SendIO serve invece per le richieste di I/O asincrono, con le quali il task continua la sua esecuzione senza entrare in stato d'attesa. Le richieste di questo tipo vengono accodate insieme alle altre, e le routine interne del dispositivo Timer le restituiscono una per una alla reply port del task, inviando ogni volta un segnale per avvertire il task.

Il task può creare tante reply port quante gliene servono. Se, per esempio, deve effettuare tre temporizzazioni, può creare tre reply port e inizializzare tre strutture timerequest (la struttura di I/O prevista dal dispositivo Timer) per altrettanti comandi TR_ADDREQUEST, specificando per ognuna un differente parametro mn_ReplyPort (cioè, una differente reply port). Come di consueto, il task può utilizzare GetMsg oppure Remove per rimuovere i messaggi restituiti nelle code alle reply port in seguito a comandi asincroni.

Nel dispositivo Timer la coda alla request port ha un'organizzazione diversa da quella esistente negli altri dispositivi. Le richieste relative a comandi TR_ADDREQUEST (il comando che ordina al dispositivo di restituire la risposta dopo l'intervallo di tempo indicato) vengono infatti ordinate secondo i periodi di tempo specificati nelle rispettive strutture timerequest. Per esempio, se un task invia uno di seguito all'altro tre comandi TR_ADDREQUEST impiegando tre diverse strutture di I/O, in cui la prima indica un tempo di 5 secondi, la seconda di 10 e la terza di 2, queste richieste vengono ordinate nella coda all'unità in modo che quella da 2 secondi sia la prima, quella da 5 la seconda e quella da 10 la terza. Questo procedimento assicura che i comandi TR_ADDREQUEST indicanti intervalli di tempo più

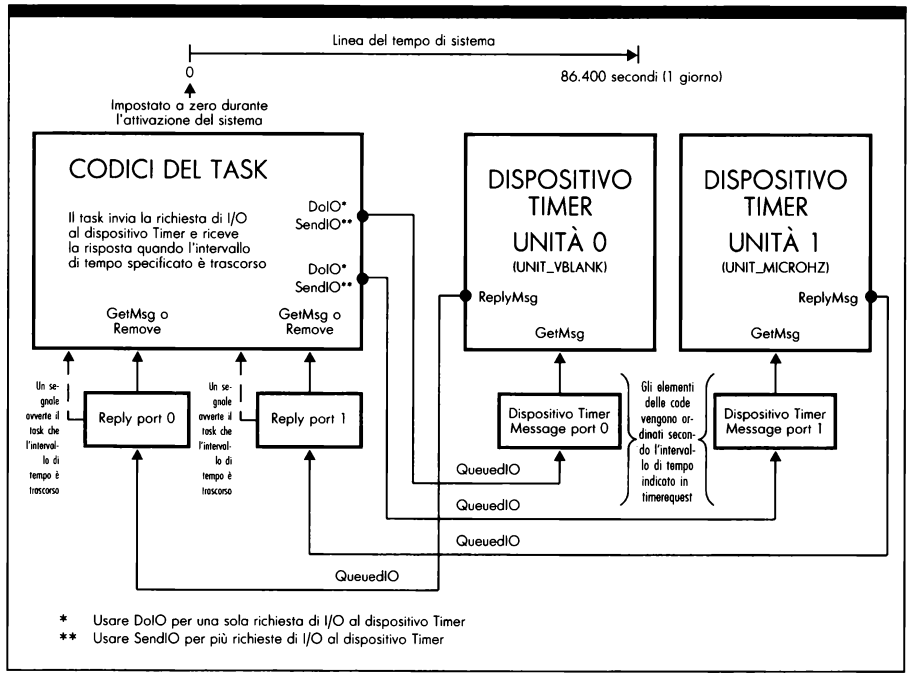


Figura 13.1: *Funzionamento del dispositivo Timer*

brevi, e quindi più urgenti, siano elaborati per primi. L'ordinamento della lista viene mantenuto aggiornato dalle routine interne del dispositivo Timer.

Oltre ai segnali di temporizzazione che i task possono richiedere tramite il comando TR_ADDREQUEST, il dispositivo Timer interagisce attraverso i comandi TR_GETSYSTIME e TR_SETSYSTIME con un particolare orologio di sistema (questo orologio è disponibile solo ai task, ai dispositivi e al sistema; non ha niente a che vedere con l'orologio che l'utente può leggere e alterare tramite i comandi dell'AmigaDOS). Questo orologio di sistema mantiene aggiornato un valore numerico, detto tempo di sistema, incrementandolo a ogni interrupt di vertical-blanking e ogni volta che un task vi accede in lettura, garantendo quindi l'assoluta unicità del valore temporale fornito. Si tratta di un valore, espresso in secondi e microsecondi, che può variare da 0 fino a un massimo di 86.400 secondi, la durata di un giorno. All'accensione della macchina questo valore viene impostato a 0.

Le funzioni AddTime, CmpTime e SubTime del dispositivo Timer permettono a un task di sommare, sottrarre e confrontare rapidamente intervalli di tempo espressi, tramite la struttura timeval, in secondi e microsecondi.

Le unità del dispositivo Timer

Come viene mostrato nella Figura 13.1 (nella pagina precedente), il dispositivo Timer possiede due unità: UNIT_MICROHZ e UNIT_VBLANK. La prima utilizza il timer programmabile contenuto nel chip 8250 CIA (Complex Interface Adapter). Nell'Amiga ne sono presenti due: 8250 CIA A e B. I registri dell'interfaccia A si trovano in RAM alle locazioni BFE001-BFEF01; i registri dell'interfaccia B si trovano agli indirizzi BFD000-BFDF00. Quattro registri dell'interfaccia A (BFE401-BFE701) e quattro dell'interfaccia B (BFD400-BFD700) sono dedicati ai calcoli dei tempi. Si veda la mappa di memoria del sistema nell'*Amiga ROM Kernel Reference Manual: Exec* per maggiori informazioni su queste interfacce e sui rispettivi registri.

UNIT_MICROHZ è in grado di scandire il tempo in microsecondi, cioè, il suo contatore interno viene incrementato ogni microsecondo. Tuttavia, questa unità tende a commettere errori di conteggio quando il sistema è particolarmente impegnato, errori che si manifestano con una certa evidenza su intervalli di tempo dell'ordine dei secondi.

UNIT_VBLANK utilizza invece l'interrupt di vertical-blanking (l'interrupt generato ogni volta che lo schermo entra in blank, cioè ogni volta che il pennello elettronico completa un quadro e si appresta a iniziarne uno nuovo) per incrementare il contatore. Nella versione americana dell'Amiga, il contatore viene incrementato 60 volte al secondo, cioè ogni 16,67 millisecondi; questo numero rappresenta anche la precisione di questa seconda unità (cioè, l'errore che l'unità può commettere rispetto all'intervallo di tempo richiesto è al massimo di 16,67 millisecondi). Nella versione europea il contatore viene incrementato 50 volte al secondo, cioè ogni 20 millisecondi; anche in questo caso, 20 rappresenta la precisione dell'unità (cioè, l'errore che l'unità può commettere rispetto all'intervallo di tempo richiesto è al massimo di 20

millisecondi). L'intervallo di tempo richiesto e l'intervallo di tempo effettivo possono quindi differire di questo scarto qualora il primo non sia un multiplo esatto della più piccola unità di tempo che il timer è in grado di gestire. Questo errore dipende dalla stessa natura dell'unità, e non dal fatto che il sistema sia impegnato nella gestione di un elevato numero di task: per evitarlo è sufficiente indicare valori temporali che siano multipli esatti del periodo minimo.

Le differenti caratteristiche delle due unità del dispositivo Timer determinano il modo in cui un task deve specificare un intervallo di tempo. UNIT_MICROHZ può misurare tempi dell'ordine del microsecondo, ma risente del livello di occupazione del sistema; UNIT_VBLANK non risente del livello di occupazione del sistema, ma non può scendere al di sotto del cinquantesimo (sessantesimo) di secondo. Per esempio, se un task desidera un segnale dopo 20 secondi, può inviare un comando TR_ADDREQUEST all'unità UNIT_VBLANK impostando il parametro tv_secs con il valore 20 e tv_micro a 0. UNIT_VBLANK calcola autonomamente quanti cinquantesimi di secondo (o sessantesimi nella versione americana) devono trascorrere per arrivare a 20 secondi e invia un segnale al task quando questo intervallo di tempo è trascorso; in questo caso il task viene avvertito quando sono effettivamente trascorsi 20 secondi esatti. Se invece il task desidera un segnale dopo un tempo di 1,045 secondi, può ancora una volta utilizzare l'unità UNIT_VBLANK specificando per il parametro tv_secs il valore 1 e per il parametro tv_micro il valore 45.000. In questo caso verrà avvertito dello scadere del tempo indicato con un errore massimo di 15 millisecondi in valore assoluto, dato che l'intervallo di tempo richiesto non è un multiplo dell'unità di tempo considerata. Se i due esempi li valutiamo nel sistema americano, otteniamo lo stesso risultato per i 20 secondi, mentre nel secondo caso otteniamo un errore massimo assoluto di 5,01 millisecondi. Come si può notare, l'unica differenza che si riscontra fra il caso europeo e il caso americano è a livello di millisecondi, mentre per i secondi non si riscontra nessuna differenza.

Se però la temporizzazione dev'essere precisa al microsecondo, il task può utilizzare l'unità UNIT_MICROHZ, specificando per il parametro tv_micro un valore compreso tra 0 e 999.999. Per esempio, se il task necessita di un intervallo di tempo di 1,5543 secondi, può specificare tv_secs come 1, tv_micro come 554.300, e inviare il comando TR_ADDREQUEST a UNIT_MICROHZ. L'intervallo di tempo che effettivamente trascorre è normalmente più accurato di quello prodotto con UNIT_VBLANK... a meno che il sistema non sia particolarmente occupato.

Correzione dei tempi in un sistema fortemente occupato

Le funzioni AddTime, CmpTime e SubTime insieme con i comandi TR_SETSYSTIME e TR_GETSYSTIME permettono a un task d'indurre il dispositivo a una maggiore accuratezza dei conteggi quando viene usata l'unità UNIT_MICROHZ e il sistema si trova a livelli di occupazione elevati. Il task può ricalibrare i parametri degli intervalli di tempo che si appresta a richiedere valutando il livello di occupazione del sistema e la sua influenza sull'unità UNIT_MICROHZ. Conviene descrivere questa operazione di compensazione

con un esempio, mostrando come TR_ADDREQUEST, TR_GETSYSTIME e SubTime possano contribuire a una maggiore precisione nel conteggio dei tempi.

Si supponga che un task voglia essere avvertito dal dispositivo Timer allo scadere di ogni secondo per un periodo di un minuto. Può utilizzare UNIT_MICROHZ e inviare all'unità una serie di comandi TR_ADDREQUEST. Ognuno di questi comandi deve indicare il parametro tv_secs inizializzato a 1 e il parametro tv_micro inizializzato a 0.

Se il sistema sta interagendo esclusivamente con questo task, il contatore di UNIT_MICROHZ viene dedicato esclusivamente al conteggio dell'intervallo specificato. Ogni volta che trascorre un secondo, il dispositivo restituisce la richiesta TR_ADDREQUEST e la inserisce nella coda alla reply port del task, il quale, tramite un segnale, viene prontamente avvertito. In questo caso i conteggi degli intervalli richiesti hanno un elevato grado di accuratezza.

Se invece il sistema sta gestendo diversi task e procede a soddisfarne simultaneamente tutte le richieste di temporizzazione, UNIT_MICROHZ non ha sempre il tempo sufficiente per incrementare il contatore ogni microsecondo. Anche se il primo segnale riuscisse a giungere al task dopo un secondo esatto, quelli successivi tenderebbero ad arrivare sempre più in ritardo. L'errore nella temporizzazione globale aumenta a mano a mano che passa il tempo.

Un task, non potendo evitare questi errori introdotti nell'unità UNIT_MICROHZ dal cosiddetto overhead di sistema, deve provvedere a compensarli. È possibile attuare questa operazione applicando la seguente procedura:

1. Il task invia tramite la funzione asincrona SendIO il primo comando TR_ADDREQUEST, che rappresenta un periodo di tempo pari a un secondo. Subito dopo deve inviare un comando TR_GETSYSTIME tramite DoIO, che richiede automaticamente il QuickIO, e salvare il tempo di sistema che ottiene (tv_secs, tv_micro) in una struttura timeval locale (Time1).

2. Il task deve quindi entrare in attesa della risposta all'ultimo comando TR_ADDREQUEST inviato a UNIT_MICROHZ; quando la ottiene deve entrare in un ciclo che per 59 volte reinvia il comando TR_ADDREQUEST ed entra in attesa della risposta (si noti che prima di ogni invio il task deve riaggiornare i parametri tv_secs e tv_micro, dal momento che il comando TR_ADDREQUEST li modifica). A livello teorico, al termine di questo ciclo dovrebbe essere trascorso esattamente un minuto. Quando il ciclo è terminato, il task invia immediatamente un altro comando TR_GETSYSTIME tramite DoIO, e salva nuovamente il tempo di sistema ottenuto (tv_secs, tv_micro) in una seconda struttura timeval locale (Time2).

3. A questo punto, i dati di cui il task è in possesso sono la durata dell'intervallo di tempo che dovrebbe essere trascorso (60 secondi), e il tempo effettivo che il dispositivo Timer ha impiegato per conteggiarlo, computabile sottraendo il secondo tempo di sistema dal primo

(Time2 – Time1). Per effettuare questa operazione, il task deve quindi utilizzare la funzione SubTime e confrontare con 60 la differenza ottenuta. Se il sistema era fortemente occupato durante il conteggio, il tempo effettivamente trascorso potrebbe essere molto superiore. Maggiore è il livello di occupazione del sistema e maggiore risulterà questa differenza. Da un punto di vista più analitico, chiamando ΔT la differenza (Time2 – Time1) e calcolando il rapporto ΔT/60, si ottiene il cosiddetto errore relativo. Se il sistema si fosse dedicato esclusivamente alla gestione del task, ΔT sarebbe prossimo a zero, e a maggior ragione lo sarebbe l'errore relativo: in un caso simile possiamo essere certi che i tempi sono stati conteggiati con notevole accuratezza. Ai fini del nostro esempio, supponiamo che il tempo effettivo sia stato di 70 secondi.

4. Supponiamo ora che il livello di occupazione, almeno per un breve periodo, rimanga pressoché costante. Sulla base di questa previsione e tenendo conto dell'errore relativo rilevato, il task, avendo bisogno di altre 60 temporizzazioni di un secondo ciascuna, invia nuovamente 60 comandi TR_ADDREQUEST, specificando però il valore 857.142 nel parametro tv_micro e 0 nel parametro tv_secs. In questo modo, quasi per assurdo, il task desidera ancora 60 segnali in un minuto, ma richiede 60 intervalli di tempo da 857.142 microsecondi, cioè circa 51 secondi.

Questo tipo di compensazione può essere effettuato periodicamente, al fine di regolare i parametri di conteggio secondo le variazioni del livello di occupazione del sistema. Il risultato, specialmente se la compensazione è molto frequente, è un'elevata accuratezza che non è influenzata dal livello di occupazione del sistema. Ma bisogna stare attenti, in quanto troppe compensazioni possono avere l'effetto opposto. Anche se teoricamente una maggiore frequenza delle compensazioni aumenta l'accuratezza, richiede anche una quantità maggiore di chiamate alla funzione TR_GETSYSTIME. Se ipoteticamente ogni comando TR_ADDREQUEST fosse preceduto e seguito da un comando TR_GETSYSTIME (massima accuratezza teorica), si otterrebbe in effetti un rallentamento maggiore, e quindi una bassa accuratezza. Inoltre, bisogna anche tenere presente che il tempo di sistema ha una precisione che non si spinge oltre il sessantesimo di secondo (un cinquantesimo nella versione europea), cosa che potrebbe introdurre macroscopici errori nella compensazione quando questa viene effettuata su periodi di tempo troppo brevi. Occorre quindi tener conto di volta in volta di tutti questi fattori e raggiungere il giusto compromesso.

Alcuni dati possono rendere più evidenti le differenze fra le due unità all'aumentare del grado di occupazione del sistema. Immaginiamo di possedere due task che inviano 60 richieste di temporizzazione, ognuna da un secondo, al dispositivo Timer. Il primo impiega l'unità UNIT_VBLANK e il secondo l'unità UNIT_MICROHZ. Nelle condizioni migliori, cioè con il sistema occupato al minimo, il primo task termina la propria esecuzione dopo 60 secondi e 102 millisecondi, mostrando un errore dello 0,17 per cento (imputabile al fatto che la CPU è di tanto in tanto impegnata in attività che non riguardano il task), mentre il secondo task mostra un errore dell'1,23 per cento. Se si rimandano in

esecuzione i task compilando contemporaneamente un programma (ovvero aumentando il grado di occupazione del sistema), il primo task mostra un errore dello 0,40 per cento, mentre il secondo un errore del 7,36 per cento. Tali risultati inducono alle seguenti considerazioni: il peggioramento dello 0,23 per cento impiegando l'unità UNIT_VBLANK è da imputare esclusivamente al fatto che nel corso del secondo test il task riceve meno partizioni di tempo della CPU, occupata anche dalla compilazione, e quindi UNIT_VBLANK non ha sofferto dell'aumento di overhead nel sistema. Ovviamente questo tipo di peggioramento influisce anche sull'esecuzione del secondo task, quello che impiega l'unità UNIT_MICROHZ, ma soltanto in minima parte dal momento che gli errori introdotti da questa unità all'aumentare del grado di occupazione del sistema hanno un peso molto più rilevante. Tenendo conto di questo 0,23 per cento comune a entrambi i test (quello che ha usato UNIT_VBLANK e quello che ha usato UNIT_MICROHZ), notiamo che in 60 secondi l'unità UNIT_MICROHZ è arrivata a commettere un errore del 7,13 per cento, peggiorando del 5,9 per cento rispetto a quando la macchina era poco occupata.

comandi del dispositivo Timer

Il dispositivo Timer non prevede alcun comando standard e possiede tre comandi specifici: TR_ADDREQUEST, TR_GETSYSTIME e TR_SETSYSTIME. Tutti e tre permettono il QuickIO, ma nessuno viene eseguito in in modo immediato. Tutti i comandi influenzano il parametro io_Error della struttura timerequest; TR_GETSYSTIME influenza anche i parametri contenuti nella struttura timeval.

L'invio dei comandi al dispositivo Timer

La Figura 13.2 (nella pagina successiva) mostra lo schema generale utilizzato per l'invio dei comandi alle routine interne del dispositivo Timer. Le linee con le frecce rappresentano i parametri che devono essere inizializzati e quelli che vengono restituiti dalle routine interne del dispositivo.

La programmazione del dispositivo Timer prevede tre fasi.

1. *Preparazione della struttura timerequest*. In questa fase il programmatore possiede il completo controllo. Qui vengono inizializzati i parametri nella struttura timerequest in preparazione dell'invio di un comando alle routine interne del dispositivo Timer. Oltre ai parametri standard ci sono i parametri tv_secs e tv_micro della struttura timeval. La scelta dei parametri dipende comunque dal comando che s'intende inviare.

2. *Elaborazione del comando da parte del dispositivo*. In questa fase l'unico compito a carico del programmatore è l'invio del comando al dispositivo utilizzando le funzioni DoIO o SendIO. Il controllo passa quindi alle routine interne del dispositivo e del sistema.

3. *Elaborazione dei parametri da restituire*. In questa fase il controllo passa completamente alle routine interne del dispositivo Timer, che restituiscono al task i risultati prodotti dall'elaborazione del comando. Se non viene concesso il QuickIO, la richiesta viene accodata alla request port, ne raggiunge la sommità e viene elaborata, quindi viene restituita alla reply port del task. Se invece il QuickIO ha successo, la richiesta viene restituita al task direttamente.

Due comandi del dispositivo Timer (TR_ADDREQUEST e TR_GETSYSTIME) forniscono parametri in uscita; tuttavia, i parametri tv_secs e tv_micro restituiti da TR_ADDREQUEST non hanno alcun significato. Tutti e tre i comandi del dispositivo Timer forniscono i risultati dell'esecuzione nel parametro io_Error.

La Figura 13.2 mostra anche i parametri che giocano un ruolo significativo nell'inizializzazione e nell'elaborazione delle funzioni del dispositivo. Timer possiede tre funzioni specifiche: AddTime, SubTime e CmpTime; tutte e tre si servono di puntatori alla struttura timeval. Le funzioni OpenDevice e CloseDevice influenzano i parametri unit_OpenCnt e lib_OpenCnt appartenenti rispettivamente alle strutture Unit e Device; OpenDevice influenza anche il parametro io_Error.

Le strutture del dispositivo Timer

Il dispositivo Timer si serve di due strutture, timerequest e timeval, come si vede nella Figura 13.3 (nella pagina successiva). La struttura timerequest viene utilizzata per formulare le richieste di I/O da inoltrare alle routine interne del dispositivo Timer. La struttura timeval rappresenta il tempo di sistema per i comandi TR_GETSYSTIME e TR_SETSYSTIME, un intervallo di tempo per il comando TR_ADDREQUEST, per le funzioni AddTime, SubTime e CmpTime.

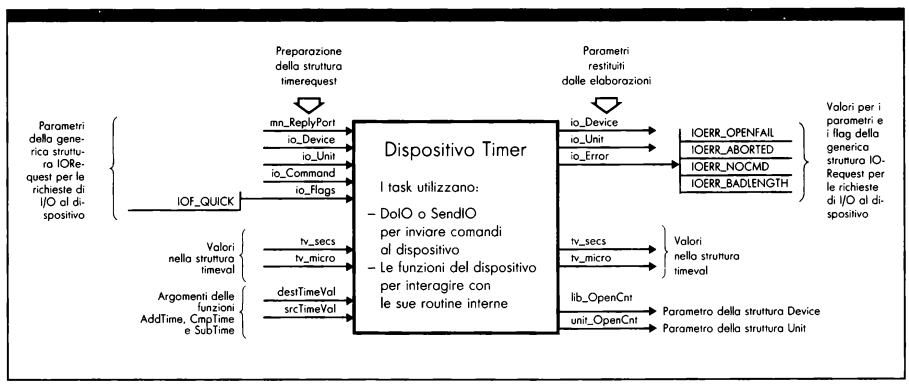


Figura 13.2: *Gestione delle funzioni e dei comandi previsti dal dispositivo Timer*

La struttura timeval

La struttura timeval è definita come segue:

```
struct timeval {
        ULONG tv_secs;
        ULONG tv_micro;
};
```

I suoi parametri indicano un periodo di tempo, e hanno il seguente significato:

- tv_secs contiene il numero di secondi.
- tv_micro rappresenta il numero di microsecondi che, insieme con i secondi del parametro tv_secs, costituiscono l'intervallo di tempo. Questo parametro può variare tra 0 e 999.999.

La struttura timerequest

La struttura timerequest è definita come segue:

```
struct timerequest {
        struct IORequest tr_node;
        struct timeval tr_time;
};
```

I suoi parametri hanno il seguente significato:

- tr_node è il nome di una sotto-struttura IORequest utilizzata per rappresentare le richieste di I/O che si inviano al dispositivo Timer.

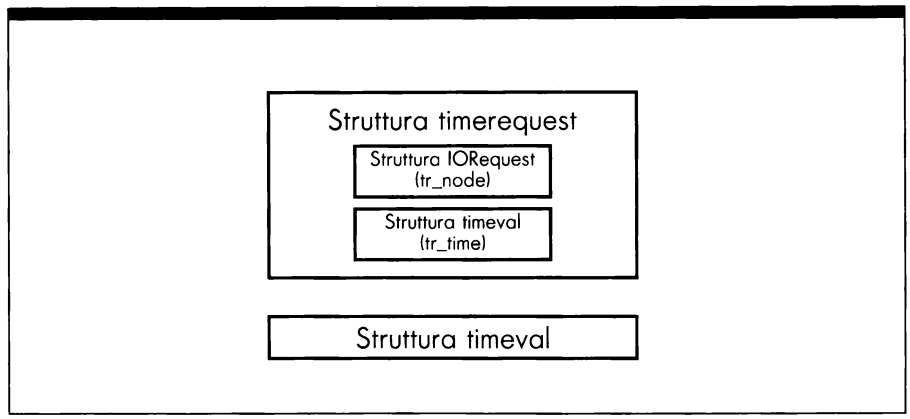


Figura 13.3: *Le strutture utilizzate dal dispositivo Timer*

I suoi parametri io_Message, io_Device, io_Unit, io_Command, io_Flags e io_Error vengono utilizzati nel modo consueto. Quando si chiamano le funzioni AddTime, SubTime e CmpTime, il parametro io_Device viene utilizzato dal sistema per ottenere l'accesso alla libreria del dispositivo Timer e calcolare gli offset di queste tre routine nella tavola dei vettori di salto.

- tr_time è il nome di una sotto-struttura timeval che con il comando TR_ADDREQUEST rappresenta l'intervallo di tempo richiesto, e con i comandi TR_SETSYSTIME e TR_GETSYSTIME rappresenta il tempo di sistema.

IMPIEGO DELLE FUNZIONI

AddTime

Sintassi di chiamata della funzione

```
AddTime (destTimeVal, srcTimeVal)
         A0           A1
```

Scopo della funzione

Questa funzione addiziona i parametri tv_micro e tv_secs di una struttura timeval a quelli di un'altra. Il risultato viene memorizzato nei parametri tv_micro e tv_secs della struttura timeval indicata nell'argomento destTimeVal.

Per accedere a questa funzione, un task deve inizializzare una variabile globale di sistema, denominata TimerBase, in modo che punti alla struttura Device di gestione del dispositivo Timer. Il task deve effettuare questa operazione aprendo il dispositivo Timer tramite la funzione OpenDevice e assegnando a TimerBase il valore contenuto nel parametro io_Device della struttura timerequest restituita dopo la prima chiamata alla funzione OpenDevice.

```
APTR TimerBase;
...
main() {
    ...
    TimerBase = (APTR)(timerequest->tr_node.io_Device);
    ...
}
```

Effettuando questa inizializzazione, la variabile TimerBase viene a contenere l'indirizzo base della libreria del dispositivo Timer. Successivamente, il task può chiamare le funzioni AddTime, CmpTime e SubTime come qualsiasi altra funzione del sistema. I registri puntatori A0 e A1 rimangono immutati al momento del ritorno della funzione AddTime.

Argomenti della funzione

destTimeVal Deve contenere l'indirizzo della struttura timeval a cui è destinato il risultato dei calcoli.

srcTimeVal Deve contenere l'indirizzo della struttura timeval che costituisce il secondo addendo dell'addizione.

Discussione

La funzione AddTime permette a un task di sommare i valori di temporizzazione rappresentati da due strutture timeval, e quindi di compiere operazioni aritmetiche sul tempo di sistema come quelle di compensazione che abbiamo visto per l'unità UNIT_MICROHZ.

Essendo una funzione, AddTime effettua un accesso diretto nella libreria di routine del dispositivo Timer. Quindi, un task non deve preparare una struttura timerequest per chiamare la funzione. Tuttavia, il task deve aver già aperto il dispositivo Timer quando effettua la chiamata alla funzione. Oltre ad aprire il dispositivo, il task deve anche inizializzare la variabile TimerBase in modo che punti alla struttura Device il cui indirizzo è stato restituito nel parametro io_Device della struttura timerequest con la prima chiamata della funzione OpenDevice.

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (timerequest)
A1

Scopo della funzione

Questa funzione chiude per un task l'accesso a un'unità del dispositivo Timer. Quando CloseDevice viene restituita, il task non può più accedere all'unità finché non esegue un'altra chiamata alla funzione OpenDevice. Tuttavia, la configurazione dei parametri interni del dispositivo viene salvata per un futuro impiego da parte di un'eventuale funzione OpenDevice inviata da questo o da un altro task.

CloseDevice decrementa il parametro lib_OpenCnt della struttura Device e il parametro unit_OpenCnt della struttura Unit.

Argomenti della funzione

timerequest	Deve contenere l'indirizzo della struttura timerequest che il task impiega per interagire con l'unità del dispositivo.

Discussione

Tramite questa funzione, un task chiude l'accesso a un'unità del dispositivo. Un task deve sempre verificare che le routine interne del dispositivo Timer abbiano restituito tutte le richieste di I/O inviate, prima di chiamare la funzione CloseDevice per chiudere l'unità. Il task può effettuare tale controllo utilizzando le funzioni GetMsg, Remove, CheckIO e WaitIO.

Il dispositivo Timer opera nel modo di accesso condiviso, e quindi non è richiesto che un task chiuda un'unità perché un altro task possa aprirla. Comunque, quando l'unità non è più necessaria conviene sempre chiuderla, se non altro per aumentare la memoria libera del sistema.

CmpTime

Sintassi di chiamata della funzione

```
result = CmpTime (firstTimeVal, secondTimeVal)
                  A0            A1
```

Scopo della funzione

Questa funzione confronta i parametri tv_micro e tv_secs di una struttura timeval con quelli di un'altra. Per utilizzare CmpTime, un task deve inizializzare una variabile globale di sistema, denominata TimerBase, in modo che punti alla struttura Device di gestione del dispositivo. Si veda la discussione della funzione AddTime per sapere come procedere.

Il risultato prodotto dall'esecuzione della funzione è espresso come valore intero, ed è restituito nella variabile result. Ecco i suoi possibili valori:

- il valore 0 indica che i parametri tv_micro e tv_secs della prima e della seconda struttura timeval sono identici.
- Il valore 1 indica che i parametri tv_micro e tv_secs della prima struttura timeval rappresentano un intervallo di tempo minore rispetto a quello rappresentato dalla seconda struttura.
- Il valore −1 indica che i parametri tv_micro e tv_secs della prima struttura timeval rappresentano un intervallo di tempo maggiore rispetto a quello contenuto nella seconda struttura.

Argomenti della funzione

firstTimeVal Deve contenere l'indirizzo della prima struttura timeval.

secondTimeVal Deve contenere l'indirizzo della seconda struttura timeval.

Discussione

La funzione CmpTime permette a un task di confrontare due temporizzazioni indicate dai parametri tv_micro e tv_secs di due diverse strutture timeval. Si veda la discussione della funzione AddTime per maggiori informazioni sull'aritmetica delle temporizzazioni.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("timer.device", unitNumber, timerequest, ØL)
DØ                 AØ               DØ          A1           D1
```

Scopo della funzione

Questa funzione apre per il task l'accesso a una specifica unità del dispositivo Timer, inizializzando i parametri io_Device e io_Unit della struttura di I/O indicata come argomento. Inoltre, OpenDevice incrementa il parametro lib_OpenCnt della struttura Device e il parametro unit_OpenCnt della struttura Unit.

OpenDevice richiede una reply port inizializzata in modo appropriato, e un bit di segnale ivi allocato dal task per ricevere un segnale di avvertimento ogni volta che il dispositivo inoltra la risposta a una sua richiesta di I/O. I risultati prodotti dall'esecuzione della funzione sono i seguenti:

- io_Device. Contiene un puntatore alla struttura Device che contiene tutte le informazioni necessarie alla gestione del dispositivo Timer.

- io_Unit. Contiene il puntatore a una struttura Unit utilizzata per definire e gestire una struttura MsgPort per l'unità del dispositivo Timer; questa struttura MsgPort gestisce la coda alla request port dell'unità.

- io_Error. Il valore 0 indica che l'apertura del dispositivo è avvenuta senza difficoltà. IOERR_OPENFAIL indica che l'unità non può essere aperta. Di solito questo problema sorge nel caso che ci sia poca memoria libera nel sistema, oppure se la richiesta rappresenta il tentativo di aprire per la seconda volta un'unità già aperta (e mai chiusa) da parte dello stesso task.

Argomenti della funzione

"timer.device"	Deve contenere l'indirizzo di una stringa a terminazione nulla con il nome del dispositivo Timer.
unitNumber	Indica l'unità del dispositivo Timer che s'intende aprire: UNIT_VBLANK o UNIT_MICROHZ.
timerequest	Deve contenere l'indirizzo di una struttura timerequest creata e opportunamente inizializzata dal task.
ØL	Questo argomento indica che i flag sono ignorati dal dispositivo Timer.

Preparazione della struttura timerequest

Si deve inizializzare mn_ReplyPort in modo che punti a una struttura MsgPort che rappresenta la reply port del task a cui vengono restituite le strutture dei comandi dopo l'elaborazione.

Discussione

Anche se il dispositivo Timer viene aperto automaticamente dall'AmigaDOS e dai dispositivi Parallel, Serial, Console e Input, il task che possiede alcuni di questi dispositivi aperti, oppure che interagisce con l'AmigaDOS, è ugualmente necessario che proceda esplicitamente all'apertura del dispositivo Timer.

Una volta che il dispositivo Timer è stato aperto, un task può inviare una serie di comandi per ottenere le temporizzazioni del sistema e creare eventi temporizzati. Quando il task ha terminato l'interazione con il dispositivo, deve provvedere a chiuderlo tramite una chiamata alla funzione CloseDevice.

SubTime

Sintassi di chiamata della funzione

SubTime (destTimeVal, srcTimeVal)
A0 A1

Scopo della funzione

Questa funzione sottrae i valori dei parametri tv_micro e tv_secs di una struttura timeval dagli analoghi parametri di un'altra struttura timeval. Il minuendo è il valore contenuto nella prima struttura timeval (destTimeVal), mentre il sottraendo è il valore contenuto nella seconda struttura timeval (srcTimeVal). Il risultato viene memorizzato nei parametri tv_micro e tv_secs della prima struttura. Dato che questa funzione fa parte delle routine interne del dispositivo Timer, un task deve inizializzare una variabile globale di sistema denominata TimerBase in modo che punti alla struttura Device che gestisce il dispositivo. Si veda la discussione della funzione AddTime per comprendere il modo in cui si svolge l'operazione.

Argomenti della funzione

destTimeVal Deve contenere l'indirizzo della struttura timeval nella quale è contenuto il minuendo della sottrazione; la funzione memorizza il risultato in questa struttura.

srcTimeVal Deve contenere l'indirizzo della struttura timeval nella quale è contenuto il sottraendo della sottrazione.

Discussione

La funzione SubTime permette a un task di sottrarre i valori di temporizzazione rappresentati da due strutture timeval, e quindi di compiere operazioni aritmetiche sul tempo di sistema. Si veda anche la discussione della funzione AddTime.

COMANDI SPECIFICI DEL DISPOSITIVO

TR_ADDREQUEST

Scopo del comando

TR_ADDREQUEST permette a un task di temporizzare le proprie operazioni. Viene inoltrato alle routine interne del dispositivo Timer per effettuare un conto alla rovescia. Quando la richiesta viene inoltrata all'unità, il dispositivo la inserisce nella relativa coda tenendo conto della sua urgenza rispetto alle altre richieste eventualmente presenti. Ovviamente, più l'intervallo di tempo è breve e più aumenta l'urgenza della corrispondente richiesta di I/O.

Quando l'intervallo di tempo specificato è interamente trascorso, l'unità restituisce la struttura della richiesta nella coda alla reply port del task. Se il task ha inviato la richiesta tramite la funzione sincrona DoIO, entra automaticamente in attesa e riottiene il controllo solo allo scadere del tempo richiesto. Se invece ha impiegato la funzione asincrona SendIO, viene avvisato dell'arrivo della risposta alla sua reply port tramite un segnale.

TR_ADDREQUEST permette il QuickIO e viene restituita alla reply port del task soltanto se il QuickIO non è stato accordato. Il valore 0 nel parametro io_Error indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato non correttamente.

I parametri tv_micro e tv_secs della struttura timerequest non contengono alcun valore significativo una volta che il comando è stato eseguito. Quindi, se si desidera utilizzare la struttura timerequest per un altro comando TR_ADDREQUEST, occorre inizializzare nuovamente i due parametri della struttura timeval prima d'inviare il nuovo comando. Il task deve assegnare uno dei suoi bit di segnale alla reply port se desidera essere avvisato quando la struttura di gestione del comando TR_ADDREQUEST viene restituita. Quando giunge la risposta, il task può essere certo che il tempo desiderato è trascorso.

Preparazione della struttura timerequest

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura timerequest inizializzata con la prima chiamata ålla funzione OpenDevice. Si devono inizializzare infine i seguenti parametri:

io_Command a TR_ADDREQUEST, io_Flags a IOF_QUICK per richiedere il QuickIO (altrimenti a 0), tv_secs con il numero di secondi contenuti nell'intervallo di tempo desiderato e tv_micro con il numero di microsecondi da sommare ai secondi per ottenere l'esatto intervallo di tempo.

Discussione

Quando un task utilizza la funzione SendIO per inviare una serie di comandi TR_ADDREQUEST, non entra automaticamente in stato di attesa, e può svolgere altre mansioni. Sono le routine interne del dispositivo Timer che avvisano il task (tramite segnali) quando l'intervallo di tempo specificato è trascorso (ovviamente il task deve preoccuparsi di entrare in attesa tramite la funzione Wait). Una volta avvisato, il task può utilizzare le funzioni CheckIO, WaitIO e GetMsg per elaborare le risposte presenti nella propria reply port. Se un task utilizza invece la funzione DoIO per inviare un comando TR_ADDREQUEST, rimane bloccato in attesa fino a quando l'intervallo di tempo specificato non è trascorso.

L'ordine in cui vengono inviati i comandi TR_ADDREQUEST al dispositivo non ha alcuna importanza; le routine interne effettuano periodicamente l'ordinamento delle strutture timerequest presenti nella coda dell'unità, dando sempre la precedenza alle richieste di I/O che indicano gli intervalli di tempo più brevi.

TR_ADDREQUEST può essere inviato richiedendo il QuickIO. Con il QuickIO, le routine interne del dispositivo Timer non inviano alcun segnale al task che ha inoltrato il comando. Quindi, il miglior impiego del QuickIO si ha quando il task deve attendere che trascorra un tempo preciso, e intanto non deve eseguire alcuna operazione: è sufficiente chiamare la funzione DoIO, che richiede il QuickIO automaticamente, e il task riprenderà l'esecuzione della successiva istruzione del programma appena sarà trascorso il tempo indicato.

TR_GETSYSTIME

Scopo del comando

Questo comando permette a un task di ottenere informazioni sul tempo di sistema. Il tempo di sistema viene inizializzato a 0 al momento dell'attivazione della macchina e raggiunge un massimo di 86.400 secondi (tanti quanti ne sono contenuti in un giorno); raggiunto questo valore, viene nuovamente azzerato. Il tempo di sistema viene incrementato di un cinquantesimo di secondo (un sessantesimo nei sistemi americani) ogni volta che si verifica un interrupt di vertical-blanking (l'interrupt generato ogni volta che il pennello elettronico che

scandisce lo schermo completa un quadro e si appresta a iniziarne uno nuovo). Quando si inoltra il comando si ottiene un valore espresso in secondi e microsecondi. Il tempo di sistema viene inoltre incrementato automaticamente ogni volta che un task inoltra il comando TR_GETSYSTIME, in modo che il valore ottenuto costituisca sempre un riferimento unico e irripetibile.

TR_GETSYSTIME permette il QuickIO e viene quindi restituito alla reply port del task soltanto se il QuickIO non ha successo. Il risultato prodotto dall'esecuzione del comando, cioè il tempo di sistema, viene restituito nei parametri tv_secs (secondi) e tv_micro (microsecondi) della struttura timeval individuata dalla struttura timerequest. Il parametro io_Error contiene il valore 0 se il comando è stato eseguito; il valore IOERR_NOCMD indica invece che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura timerequest

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura timerequest inizializzata con la prima chiamata alla funzione OpenDevice. Si devono inizializzare infine io_Command a TR_GETSYSTIME, io_Flags a IOF_QUICK (per richiedere il QuickIO) e tv_secs e tv_micro a 0.

Discussione

I comandi TR_GETSYSTIME e TR_SETSYSTIME dovrebbero essere sempre inviati richiedendo il QuickIO, cioè impiegando la funzione DoIO. Senza il QuickIO la richiesta verrebbe accodata nella coda alla request port dell'unità, e verrebbe poi inserita (a elaborazione ultimata) nella reply port del task. L'elaborazione delle code richiede tempo, cosa che non sarebbe positiva per due comandi che devono rispettivamente riportare e impostare il tempo di sistema.

TR_SETSYSTIME

Scopo del comando

Questo comando permette a un task d'inizializzare il tempo di sistema a un particolare valore. Il tempo di sistema viene automaticamente inizializzato a 0 al momento dell'attivazione della macchina e viene incrementato fino a 86.400 secondi (i secondi contenuti in un giorno), dopodiché riparte da 0. È espresso in secondi e microsecondi, ma viene aggiornato a ogni interrupt di vertical-blanking (quello generato ogni volta che il pennello elettronico che scandisce lo schermo completa un quadro e si appresta a iniziarne uno nuovo). Questo interrupt si verifica ogni sessantesimo di secondo nei sistemi americani, e ogni cinquantesimo di secondo nei sistemi europei.

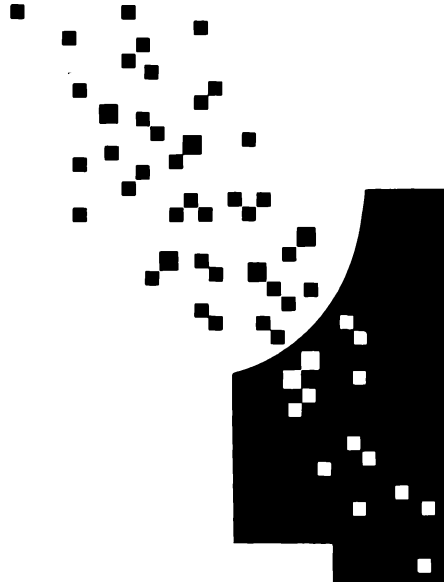
TR_SETSYSTIME consente il QuickIO e viene restituito alla reply port del task soltanto se il QuickIO non viene accordato. I risultati prodotti dall'esecuzione del comando sono restituiti nel parametro io_Error. Il valore 0 indica che il comando è stato eseguito. IOERR_NOCMD indica che il parametro io_Command è stato specificato in modo non corretto.

Preparazione della struttura timerequest

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono il dispositivo e l'unità; questi parametri devono essere copiati dalla struttura timerequest inizializzata con la prima chiamata alla funzione OpenDevice. Si devono inizializzare infine io_Command a TR_SETSYSTIME, io_Flags a IOF_QUICK (per richiedere il QuickIO), tv_secs con il numero di secondi da assegnare al tempo di sistema e tv_micro con il numero dei microsecondi.

Discussione

Il comando TR_SETSYSTIME permette a un task d'inizializzare il tempo di sistema a un particolare valore, in modo da soddisfare le proprie necessità di temporizzazione. Come per il comando TR_GETSYSTIME, TR_SETSYSTIME dovrebbe sempre essere inviato richiedendo il QuickIO, cioè tramite la funzione DoIO.



14

Il dispositivo TrackDisk

Introduzione

Il dispositivo TrackDisk controlla i quattro disk drive di cui un Amiga può essere dotato. Viene utilizzato per scrivere e leggere dati nelle tracce del disco (sia grezzi sia codificati) per ottenere informazioni sullo stato dei disk drive, e per aggiungere al sistema routine di interrupt adibite al controllo delle sostituzioni dei dischi (estrazioni e inserimenti). Il dispositivo TrackDisk è l'interfaccia software con il sistema dei dischi di più basso livello, e viene utilizzato anche dall'AmigaDOS. Risiede su ROM, e nel caso dell'Amiga 1000 viene caricato automaticamente dal disco del Kickstart nella ROM WCS durante l'attivazione del sistema.

Alcuni comandi standard dei dispositivi non sono previsti dal dispositivo TrackDisk, che comunque fornisce dei comandi specifici equivalenti. Per quanto riguarda le strutture di I/O, questo dispositivo prevede l'impiego di tre strutture: IOExtTD, TDU_PublicUnit e BootBlock.

Funzionamento del dispositivo TrackDisk

La Figura 14.1 (nella pagina successiva) illustra il funzionamento generale del dispositivo, il quale è in grado di accedere sia ai disk drive per dischi da 3,5" sia a quelli per dischi da 5,25".

Possiede quattro unità che possono essere aperte solo nel modo di accesso condiviso. L'unità 0 è dedicata al disk drive di serie; le unità 1, 2 e 3 sono dedicate rispettivamente al primo, al secondo e al terzo disk drive eventualmente aggiunti alla configurazione base. I disk drive aggiuntivi vengono fisicamente disposti in cascata. Il connettore al quale si può collegare il primo dei disk drive esterni si trova sul retro dell'Amiga (nell'Amiga 2000 vi sono due disk drive interni e quindi, per questa macchina, il primo disk drive esterno coincide con il secondo disk drive aggiuntivo). La Tavola 14.1 (a pagina 439) riassume le funzioni dei vari pin del connettore.

Il dispositivo TrackDisk è orientato alla gestione dei dati in blocchi; ogni accesso al disco comporta il trasferimento di un'intera traccia (o cilindro). Il QuickIO, nel caso degli accessi al disco non aumenterebbe la velocità di risposta del sistema, e pertanto non viene utilizzato. Le richieste di I/O vengono quindi accodate nelle code alle request port delle varie unità.

Ogni unità possiede un buffer, detto buffer di traccia, in grado di contenere i dati di tutti gli 11 settori che costituiscono una traccia. I buffer di traccia sono gestiti dalle routine interne del dispositivo TrackDisk, e quindi i task possono controllarli solo in parte. Questi buffer agiscono come locazioni intermedie per i dati grezzi o pre-elaborati in transito tra i buffer dei task e i disk drive. Tutti i buffer per i trasferimenti di dati definiti dai task devono trovarsi nella chip RAM (attualmente limitata ai primi 512K di memoria) e devono iniziare allineandosi alle word.

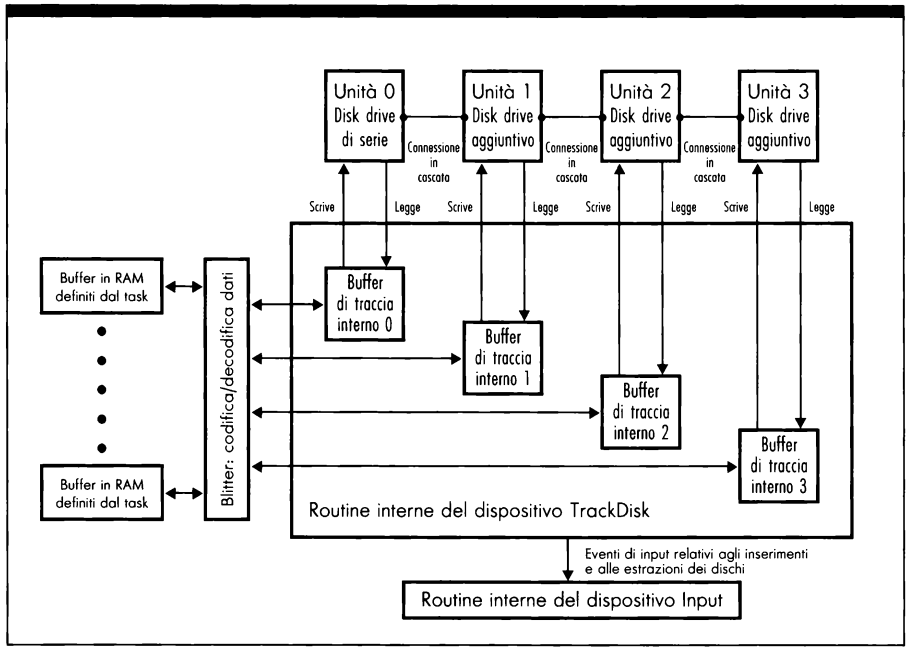


Figura 14.1: *Funzionamento del dispositivo TrackDisk*

Le operazioni di lettura (effettuabili tramite i comandi TD_RAWREAD, ETD_RAWREAD, ETD_READ e CMD_READ) trasferiscono nel buffer di traccia dell'unità i dati contenuti in un'intera traccia del disco, e successivamente trasferiscono nel buffer definito dal task i dati che erano stati richiesti; se si usa il comando ETD_READ (o il comando CMD_READ) durante il trasferimento i dati vengono decodificati dal coprocessore Blitter in un apposito formato interno. Le operazioni di scrittura (effettuabili tramite i comandi TD_RAWWRITE, ETD_RAWWRITE e ETD_WRITE e CMD_WRITE) trasferiscono uno o più settori di dati (per i comandi grezzi un'intera traccia) dal buffer interno del task al buffer di traccia dell'unità e successivamente al disco; se si usa il comando ETD_WRITE (o il comando CMD_WRITE), i dati vengono codificati dal coprocessore Blitter.

Il dispositivo TrackDisk, inoltre, invia al dispositivo Input gli eventi di input generati dall'inserimento e dalla rimozione dei dischi. Un task può utilizzare il comando TD_ADDCHANGEINT per installare nel sistema una routine di interrupt software da eseguire quando un disco viene inserito o rimosso. Si veda il capitolo 7 per maggiori informazioni sul dispositivo Input e sulle sue interazioni con TrackDisk.

Pin	Data	Direzione	Descrizione
1	RDY	Input/output	Disco inserito (se motore acceso) o identificazione
2	DKRD	Input	Dati MFM (modified frequency modulation) in input all'Amiga
3-7	GND		Massa
8	MTRXD	Collettore aperto	Motore acceso/spento
9	SEL2B	Collettore aperto	Seleziona disk drive 2
10	DRESB	Collettore aperto	Reset di sistema dall'Amiga
11	CHNG	Input/output	Toggle per disco inserito/ non inserito
12	+5 volt		270 mA massimi; 410 mA di picco
13	SIDEB	Output	Lato 1 se attivo, lato 0 se inattivo
14	WPRO	Input/output	Disco protetto in scrittura
15	TK0	Input/output	Testina di lettura-scrittura su traccia 0
16	DKWEB	Collettore aperto	Porta di scrittura al disk drive
17	DKWDB	Collettore aperto	Dati MFM in output dall'Amiga
18	STEPB	Collettore aperto	Il disk drive selezionato muove la testina di una traccia nella direzione DIRB
19	DIRB	Collettore aperto	Direzione di movimento della testina
20	SEL3B	Collettore aperto	Seleziona disk drive 3
21	SEL1B	Collettore aperto	Seleziona disk drive 1
22	INDEX	Input/output	Impulso indice, uno per rotazione completa
23	+12 volt		160 mA massimi; 540 mA di picco

Tavola 14.1: *Connessioni dei pin per i disk drive esterni*

Interazioni tra disco e dispositivo TrackDisk

La Figura 14.2 (nella pagina successiva) illustra le operazioni compiute dal dispositivo TrackDisk su un disco. Ogni disco da 3,5" possiede due facce, 80 tracce per faccia, 11 settori per traccia e 512 byte per ogni settore. Il numero delle tracce varia da 0 (la traccia più esterna) a 79 (la traccia più interna). Inoltre, ogni settore contiene 16 byte che lo identificano. Il disco da 3,5" può quindi contenere 880K d'informazioni (440K per faccia), e 28K di dati per l'identificazione dei settori. Dischi formattati da altri computer possono ovviamente possedere caratteristiche diverse.

Alcune operazioni del dispositivo TrackDisk influenzano le informazioni contenute nel disco, nel buffer di traccia o in quello del task: sono le operazioni di lettura, scrittura, aggiornamento e formattazione.

- Ogni operazione di lettura o scrittura trasferisce uno o più settori tra il disco e il buffer definito dal task; il numero dei settori viene controllato dalle istruzioni del task che definiscono il trasferimento dei dati. Se un task impartisce un comando per leggere un settore da un disco (accede

cioè all'unità corrispondente al disk drive che contiene quel disco) e questo settore si trova già nel buffer di traccia dell'unità (grazie a precedenti accessi alla stessa traccia), non si verifica alcuna attività di lettura dal disco. Se invece il settore di dati richiesto non si trova nel buffer di traccia dell'unità, l'intera traccia che lo contiene viene automaticamente trasferita dal disco al buffer dell'unità. Se però il buffer contiene una traccia precedentemente alterata e non ancora riscritta su disco, l'unità del dispositivo, prima di procedere alla lettura della nuova traccia, provvede a trasferire su disco quella vecchia. Allo stesso modo, un'operazione di scrittura non avviene immediatamente se nel buffer è presente una precedente traccia già elaborata e da salvare su disco.

Gli accessi al disco in lettura e in scrittura avvengono soltanto se i dati contenuti in quel momento nel buffer di traccia non vanno perduti durante l'operazione. Se nel buffer di traccia si trova un settore in attesa di essere trasferito su disco, il dispositivo si preoccupa sempre di salvaguardarlo. Questa particolare gestione per tracce svolta dal dispositivo TrackDisk minimizza il numero di accessi fisici al disco.

- Le operazioni di aggiornamento (ETD_CLEAR, CMD_CLEAR, ETD_UPDATE e CMD_UPDATE) permettono di trasferire su disco il contenuto di tutti i buffer di traccia in situazioni di emergenza, come per esempio nel caso di una caduta di tensione.

- Le operazioni di formattazione (TD_FORMAT e ETD_FORMAT) per i

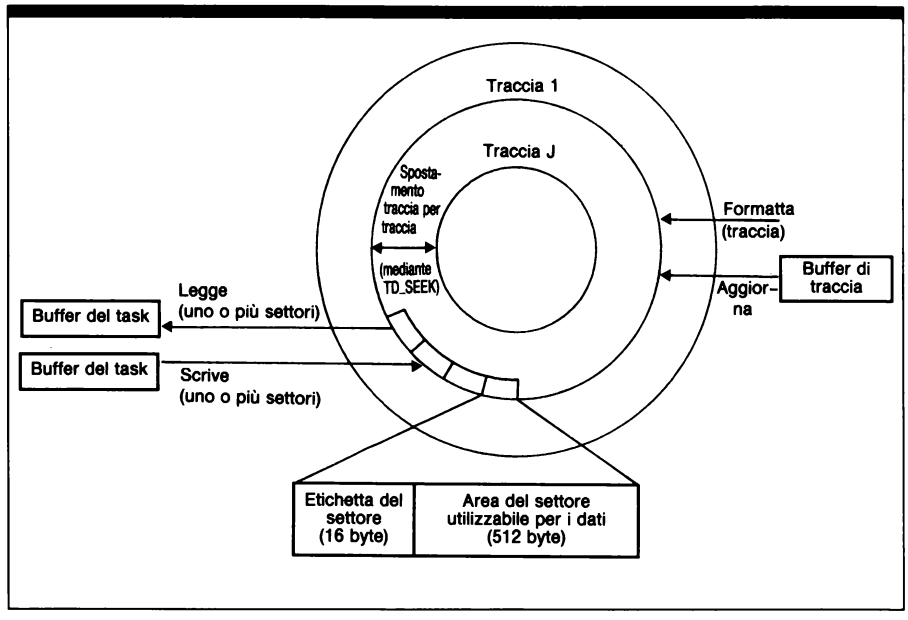


Figura 14.2: *Interazioni fra un disco e il dispositivo TrackDisk*

dischi vengono effettuate traccia per traccia, incrementando così la velocità di formattazione.

Oltre ai comandi citati, la figura illustra anche il comando TD_SEEK, con il quale è possibile muovere la testina di lettura e scrittura da traccia a traccia.

comandi del dispositivo TrackDisk

I comandi del dispositivo TrackDisk possono essere suddivisi in due categorie: i comandi "normali" e i comandi "estesi". I comandi estesi (i cui nomi sono preceduti dal prefisso ETD_) eseguono le stesse operazioni compiute dai corrispondenti comandi normali (prefisso TD_ per quelli specifici del dispositivo e CMD_ per quelli standard), con la differenza che tengono anche conto delle informazioni generate dalla sostituzione del disco o contenute nelle etichette dei settori a cui si accede. Come vedremo, i comandi estesi sono stati aggiunti per evitare di eseguire operazioni di lettura o scrittura sul disco sbagliato in seguito a una sostituzione.

L'invio dei comandi al dispositivo TrackDisk

Le Figure 14.3a e 14.3b (nelle pagine successive) descrivono lo schema generale utilizzato per inviare i comandi alle routine del dispositivo TrackDisk. Nella Figura 14.3a le linee con le frecce rappresentano i parametri da inizializzare, mentre nella Figura 14.3b rappresentano i parametri restituiti dalle routine interne del dispositivo TrackDisk. La programmazione del dispositivo TrackDisk prevede tre fasi.

1. *Preparazione delle strutture*. In questa fase il task possiede il completo controllo. Qui vengono inizializzati i parametri della struttura IOExtTD per l'invio di un comando all'unità o per l'esecuzione di una funzione del dispositivo. Oltre ai parametri utilizzati dalla maggior parte dei dispositivi ve ne sono due, iotd_SecLabel e iotd_Count, utilizzati specificamente dai comandi estesi. In particolare, se il task desidera impiegare questi comandi deve memorizzare nel parametro iotd_Count il massimo numero accettabile d'inserimenti e rimozioni del disco.

2. *Elaborazione delle richieste di I/O*. L'unico compito a carico del programmatore in questa fase è l'invio della richiesta al dispositivo utilizzando le funzioni BeginIO, DoIO o SendIO. Il controllo passa quindi alle routine interne del dispositivo.

3. *Elaborazione della richiesta e dei parametri da restituire*. Questa fase è completamente controllata dalle routine interne del dispositivo TrackDisk. La richiesta di I/O viene elaborata nel momento in cui raggiunge la sommità della coda alla request port dell'unità e i risultati

prodotti dall'elaborazione del comando vengono restituiti alla reply port del task che l'aveva inviato.

I comandi TD_GETDRIVETYPE, TD_GETNUMTRACKS, TD_CHANGENUM forniscono un risultato nel parametro io_Actual, e tutti i comandi restituiscono nel parametro io_Error l'esito dell'operazione.

Le Figure 14.3a e 14.3b descrivono inoltre i parametri che giocano una ruolo significativo nella definizione e nell'elaborazione delle funzioni nel dispositivo TrackDisk. Le funzioni OpenDevice e CloseDevice influenzano i parametri unit_OpenCnt e lib_OpenCnt appartenenti rispettivamente alle strutture Unit e Device; OpenDevice influenza anche il parametro io_Error.

Le strutture del dispositivo TrackDisk

Il dispositivo TrackDisk si serve di tre strutture non standard: IOExtTD, TDU_PublicUnit e BootBlock. Tutti i comandi normali del dispositivo TrackDisk possono anche essere inviati impiegando la struttura standard IOStdReq, mentre i comandi estesi possono essere inviati solo utilizzando la struttura IOExtTD.

La struttura TDU_PublicUnit viene utilizzata per gestire la coda alla request port di un'unità e per controllare la temporizzazione e altri aspetti delle operazioni con i dischi. La struttura BootBlock viene utilizzata per definire il

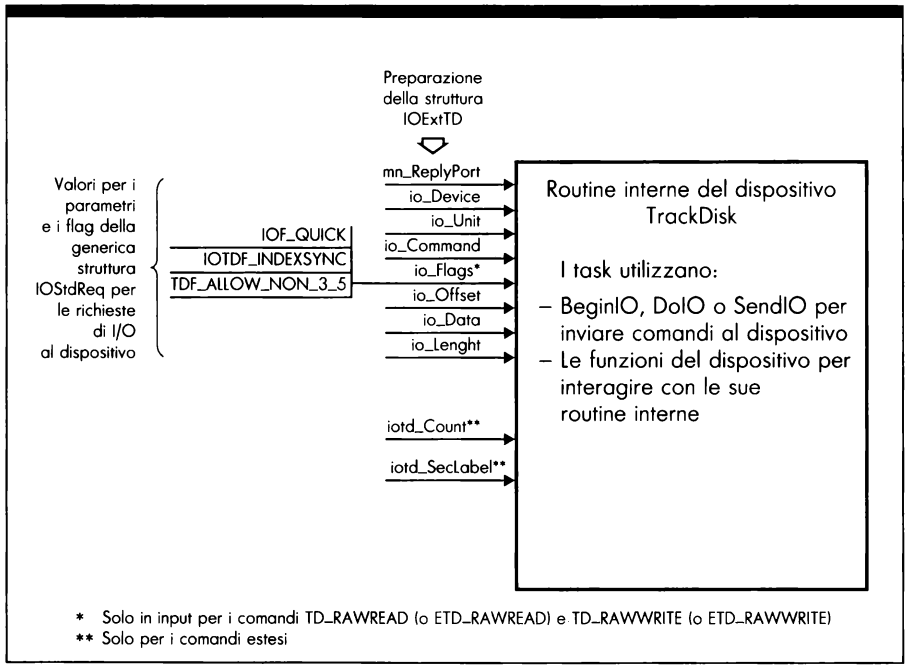


Figura 14.3a: *Gestione delle funzioni e dei comandi previsti dal dispositivo TrackDisk (input)*

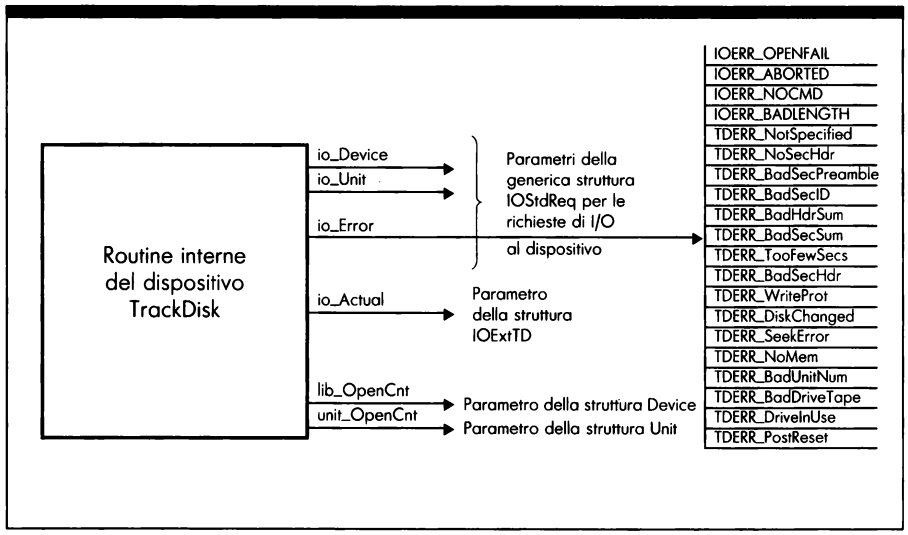


Figura 14.3b: *Gestione delle funzioni e dei comandi previsti dal dispositivo TrackDisk (output)*

boot block presente in ogni disco riconoscibile dall'Amiga. Il boot block di un disco è composto da due settori (1K). Queste strutture sono illustrate nella Figura 14.4 (nella pagina successiva).

La struttura IOExtTD

La struttura IOExtTD è definita come segue:

```
struct IOExtTD {
    struct IOStdReq iotd_Req;
    ULONG iotd_Count;
    ULONG iotd_SecLabel;
};
```

I suoi parametri hanno il seguente significato:

- iotd_Req. Questo è il nome della sotto-struttura IOStdReq tramite la quale viene gestito il transito dei messaggi, la restituzione dei codici d'errore, e viene definito il buffer che il task mette a disposizione quando inoltra un comando.

- iotd_Count. Indica al dispositivo il numero massimo d'inserimenti e rimozioni del disco che possono avvenire perché le routine interne del dispositivo non blocchino l'accesso quando si invia un comando esteso. Il task può indicare un valore qualunque, anche se generalmente indica quello che ottiene tramite il comando TD_CHANGENUM. Durante ogni accesso, il dispositivo confronta questo valore limite con il valore della

relativa variabile interna che tiene conto del numero di sostituzioni avvenute nel disk drive. Se il valore raggiunto risulta maggiore del parametro iotd_Count, le routine interne del dispositivo non elaborano la richiesta, e la restituiscono indicando nel parametro io_Error il codice d'errore TDERR_DiskChanged. In questo caso il task può desumere che dall'ultimo invio del comando TD_CHANGENUM nel disk drive prescelto l'utente ha compiuto almeno un'operazione d'inserimento o di estrazione del disco. Questa informazione può indurre il task a verificare che il disco presente nel disk drive sia quello giusto.

- iotd_SecLabel. Dev'essere azzerato, oppure deve contenere l'indirizzo di una serie contigua di buffer (composti da 16 byte) contenenti informazioni sulle etichette relative a ciascun settore di una traccia per le operazioni di lettura e di scrittura. Si noti che attualmente l'AmigaDOS non utilizza i campi-etichetta dei settori, e quindi il loro contenuto non viene copiato da tool di copia come DISKCOPY. Il numero di campi-etichetta che vengono trasferiti in ogni accesso dipende dal numero di settori ai quali si accede: se per esempio il task ha memorizzato nel parametro iotd_SecLabel l'indirizzo di un array definito come

  BYTE Etichette[NUMSECS][TD_LABELSIZE];

  e invia il comando ETD_READ per leggere un'intera traccia del disco, automaticamente il dispositivo TrackDisk memorizza nell'array Etichette le etichette di tutti gli 11 settori che compongono la traccia. Viceversa, se il task memorizza dati in ogni elemento dell'array e invia il comando ETD_WRITE per scrivere un'intera traccia, tutte le 11

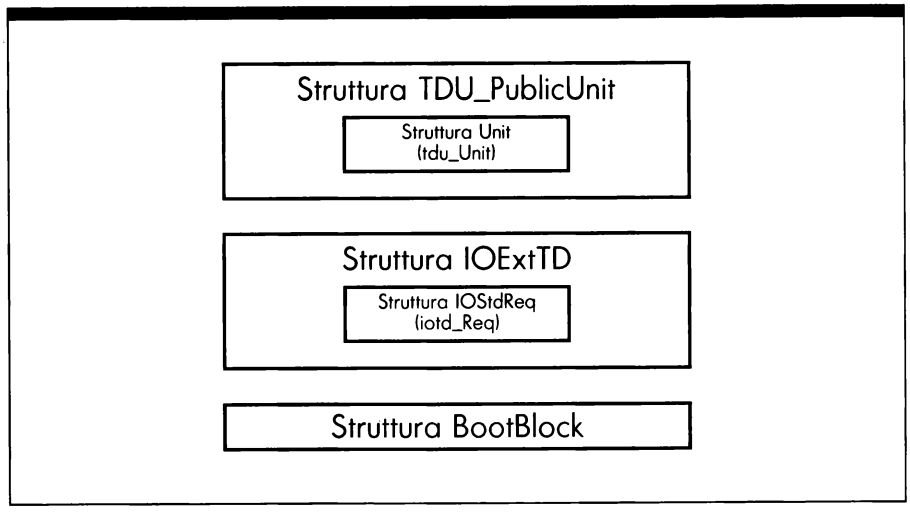


Figura 14.4: *Strutture utilizzate dal dispositivo TrackDisk*

etichette presenti in memoria vengono trasferite su disco, una per ogni settore. È opportuno ricordare che generalmente i settori dei dischi non contengono alcun dato nei loro campi-etichetta, sebbene sia molto facile accedervi sia in lettura che in scrittura. Per questa ragione, e dal momento che l'insieme dei campi-etichetta di un disco occupa ben 28.160 byte, i task che impiegano il dispositivo TrackDisk per compiere operazioni particolari con i dischi possono sfruttarli per i loro scopi. Si ricordi infine che i parametri iotd_SecLabel e iotd_Count vengono presi in considerazione solo dai comandi estesi.

Il parametro iotd_Count e i comandi estesi

Al momento dell'accensione della macchina, il dispositivo TrackDisk azzera una variabile interna per ogni disk drive collegato. In seguito, ogni volta che viene inserito o estratto un disco da un disk drive, la corrispondente variabile interna viene incrementata (se quindi si estrae e s'inserisce un disco, al termine dell'operazione la variabile risulta incrementata di due). L'incremento automatico di queste variabili avviene anche se si estrae e si rimette nel disk drive lo stesso disco, cioè non ha alcun legame con il disco che viene impiegato. Evidentemente, un valore dispari corrisponde alla situazione di disco presente nel disk drive e un valore pari corrisponde alla situazione di disk drive vuoto. I task possono accedere al valore di queste variabili interne inviando al dispositivo il comando TD_CHANGENUM, e consultando il parametro io_Actual della richiesta di I/O.

Fatte queste premesse, la ragione di esistere dei comandi estesi diventa più evidente. Se per esempio il task non desidera che l'utente estragga un disco da un disk drive durante una serie di accessi, prima d'iniziare si accerta che il disco sia presente nel disk drive (comando TD_CHANGESTATE) e invia il comando TD_CHANGENUM per ottenere il valore contenuto nella variabile interna del disk drive prescelto. Quindi lo memorizza nel parametro iotd_Count, facendo in modo di bloccare qualsiasi accesso successivo se il parametro cambia (cioè se l'utente estrae il disco). Grazie al parametro iotd_Count, i comandi estesi consentono accessi più sicuri ai dischi.

Il parametro iotd_Count può essere impiegato anche per stabilire un limite di estrazioni e d'inserimenti oltre il quale i comandi estesi non vengono più elaborati. Per esempio, se il comando TD_CHANGENUM restituisce il valore 15, il task può memorizzare nel parametro iotd_Count il valore 20 e iniziare una serie di accessi che verranno bloccati solo se l'utente estrae e inserisce il disco cinque volte.

La struttura TDU_PublicUnit

La struttura TDU_PublicUnit rappresenta una parte, quella accessibile ai programmatori, della struttura che il dispositivo impiega per gestire le unità disk drive. È definita come segue.

```
struct TDU_PublicUnit {
        struct Unit tdu_Unit;
        UWORD tdu_Comp01Track;
        UWORD tdu_Comp10Track;
        UWORD tdu_Comp11Track;
        ULONG tdu_StepDelay;
        ULONG tdu_SettleDelay;
        UBYTE tdu_RetryCnt;
};
```

I suoi parametri hanno il seguente significato:

- tdu_Unit. È la sotto-struttura Unit che il dispositivo utilizza per gestire l'unità. In essa appare una sotto-struttura MsgPort per gestire la message port dell'unità, e il parametro unit_OpenCnt, che tiene conto di quante volte l'unità è stata aperta e non ancora chiusa. Nella memoria del dispositivo, dopo la sotto-struttura tdu_Unit seguono molti altri parametri, ai quali i task possono accedere tramite la struttura TDU_PublicUnit.

- tdu_Comp01Track, tdu_Comp10Track e tdu_Comp11Track. Indicano rispettivamente le tracce da utilizzare per la prima, la seconda e la terza precompensazione di traccia.

- tdu_StepDelay. Indica il tempo (espresso in microsecondi) che il dispositivo deve attendere dopo aver richiesto al motore passo-passo uno spostamento da traccia a traccia. Nella release 1.3 del software sistema il tempo di attesa è pari a 3 millisecondi. Tuttavia, questo parametro dipende dall'hardware; un valore che risulti compatibile con il 68000 potrebbe non esserlo con il 68020, che di solito viene fatto funzionare a una frequenza di clock più elevata.

- tdu_SettleDelay. Indica il tempo (sempre espresso in microsecondi) che il dispositivo deve attendere perché la testina del disk drive si stabilizzi dopo un'operazione di spostamento. Nella release 1.3 del software sistema il tempo di attesa è pari a 15 millisecondi. Anche questo parametro dipende dall'hardware.

- tdu_RetryCnt. Indica quante volte dev'essere ritentata l'operazione di spostamento della testina (seek), qualora il disco presenti dei difetti. Nella release 1.3 del software sistema questo parametro contiene il valore 10, ma nel caso di esigenze particolari i task possono modificarlo. Se viene azzerato, i movimenti della testina diventano indifferenti ai difetti del disco.

La struttura BootBlock

La struttura BootBlock è definita come segue:

```
struct BootBlock {
      UBYTE bb_id[4];
      LONG bb_chksum;
      LONG bb_dosblock;
};
```

I suoi parametri hanno il seguente significato:

- bb_id[4]. È un identificatore composto da 4 caratteri, utilizzato per definire il tipo del disco. Attualmente, può essere inizializzato a "DOS\0" (\0 in linguaggio C significa che la stringa finisce con un byte di valore zero) per indicare un disco del DOS valido, oppure a "KICK" per indicare il disco del Kickstart.

- bb_chksum. È l'attuale valore di checksum per il boot block. Viene utilizzato per verificare l'integrità del disco al momento del boot.

- bb_dosblock. Questo parametro è riservato per futuri impieghi da parte del DOS.

Codici d'errore del dispositivo TrackDisk

I codici d'errore vengono restituiti dalle funzioni e dai comandi del dispositivo TrackDisk nel parametro io_Error della struttura IOExtTD, e hanno i seguenti significati:

- il valore 0 indica che è stato possibile eseguire il comando o la funzione senza difficoltà.

- IOERR_OPENFAIL, IOERR_ABORTED, IOERR_NOCMD e IOERR_BADLENGTH hanno gli stessi significati che assumono con gli altri dispositivi (si veda il capitolo 3).

- TDERR_NotSpecified. È un errore generico, cioè non classificabile tra quelli qui riportati.

- TDERR_NoSecHdr. Il dispositivo TrackDisk non è riuscito a identificare l'header (intestazione) di un settore.

- TDERR_BadSecPreamble. Il dispositivo TrackDisk non è riuscito a identificare il preamble (inizio) di un settore.

- TDERR_BadSecID. Un settore contiene informazioni errate nella propria etichetta.
- TDERR_BadHdrSum. È sbagliato il checksum dell'header di un settore.
- TDERR_BadSecSum. È sbagliato il checksum dei dati di un settore.
- TDERR_TooFewSecs. Il dispositivo TrackDisk non riesce a trovare abbastanza settori per soddisfare le richieste di un comando.
- TDERR_BadSecHdr. Nell'header di un settore compaiono dati errati.
- TDERR_WriteProt. Un comando ha tentato di scrivere su un disco protetto in scrittura.
- TDERR_DiskChanged. Non c'è alcun disco nel disk drive, oppure il contatore degli inserimenti e rimozioni del disk drive contiene un numero maggiore del valore indicato nel parametro iotd_Count della struttura IOExtTD.
- TDERR_SeekError. Il sistema di spostamento della testina non riesce a trovare la traccia 0 del disco.
- TDERR_NoMem. Nel sistema non risulta sufficiente memoria libera per eseguire il comando o la funzione richiesta.
- TDERR_BadUnitNum. È stato indicato un numero diverso da quelli consentiti dal sistema (0, 1, 2, 3) per indicare l'unità prescelta. Il sistema restituisce questo tipo di errore anche quando si tenta di accedere a un'unità non collegata al sistema.
- TDERR_BadDriveType. Il sistema non è in grado d'interagire con il tipo di disk drive specificato dalla funzione o dal comando.
- TDERR_DriveInUse. Il disk drive prescelto non è accessibile perché lo sta usando un altro task.
- TDERR_PostReset. L'utente ha avviato la procedura di reset del sistema impartendo la combinazione di tasti Ctrl/Amiga-Destro/Amiga-Sinistro.

IMPIEGO DELLE FUNZIONI

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iOExtTD)
A1

Scopo della funzione

Questa funzione chiude per il task l'accesso a un'unità del dispositivo TrackDisk. Quando CloseDevice restituisce il controllo, i parametri io_Device e io_Unit della struttura IOExtTD risultano impostati a −1, e per riutilizzare la struttura i due parametri devono essere nuovamente inizializzati tramite una chiamata alla funzione OpenDevice. CloseDevice decrementa il parametro lib_OpenCnt della struttura Device e il parametro unit_OpenCnt della struttura Unit.

Argomenti della funzione

iOExtTD — Deve contenere l'indirizzo della struttura di tipo IOExtTD che il task impiega per interagire con l'unità del dispositivo.

Discussione

Ogni volta che un task non ha più bisogno di un'unità del dispositivo TrackDisk deve eseguire una chiamata alla funzione CloseDevice per chiuderla.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice ("trackdisk.device", unitNumber, iOExtTD, flags)
D0                 A0                   D0          A1       D1
```

Scopo della funzione

Questa funzione apre per un task l'accesso a un'unità del dispositivo TrackDisk. OpenDevice inizializza i parametri io_Device e io_Unit della struttura IOExtTD in modo che puntino rispettivamente alle strutture Device e TDU_PublicUnit utilizzate dal sistema per gestire l'unità. OpenDevice provvede inoltre a incrementare il parametro unit_OpenCnt della struttura Unit e il parametro lib_OpenCnt della struttura Device.

Argomenti della funzione

"trackdisk.device" Deve contenere una stringa a terminazione nulla con il nome del dispositivo TrackDisk.

unitNumber Deve indicare il numero dell'unità del dispositivo TrackDisk che s'intende aprire (0L, 1L, 2L o 3L).

iOExtTD Deve contenere l'indirizzo della struttura IOExtTD creata e inizializzata dal task.

flags Dev'essere inizializzato a 0 se si desidera accedere soltanto a dischi da 3,5", oppure a TDF_ALLOW_NON_3_5 se si desidera che l'unità acceda a tutti i tipi di dischi che è in grado di gestire (normalmente l'alternativa è costituita da dischi da 5,25").

Preparazione della struttura IOExtTD

Si deve inizializzare il parametro mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task.

Discussione

Quando un'unità aperta con OpenDevice non serve più dev'essere immediatamente chiusa, al fine di risparmiare memoria e mantenere ordine nel sistema. Il task deve assegnare un bit di segnale alla reply port se vuole essere avvertito quando la richiesta di I/O effettuata con OpenDevice viene restituita.

COMANDI SPECIFICI DEL DISPOSITIVO

ETD_CLEAR, CMD_CLEAR

Scopo del comando

Questo comando identifica come non valido il contenuto del buffer di traccia dell'unità, costringendo l'unità a non impiegarlo nelle sue operazioni di accesso al disco.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che individuano l'unità del dispositivo TrackDisk; questi parametri devono essere copiati dalla struttura di I/O inizializzata con la prima chiamata a OpenDevice. Si deve inizializzare io_Command a ETD_CLEAR (o CMD_CLEAR) e io_Flags a 0. Infine, si deve memorizzare nel parametro iotd_Count il numero massimo consentito di rimozioni e inserimenti del disco, valore che può essere arbitrario oppure ottenuto tramite il comando TD_CHANGENUM. Si noti che se il task impiega il comando CMD_CLEAR, il parametro iotd_Count non viene preso in considerazione e il task può formulare la richiesta di I/O impiegando la struttura standard IOStdReq.

Discussione

I comandi ETD_CLEAR e CMD_CLEAR permettono a un task di dichiarare non valido il contenuto del buffer di traccia dell'unità. Quest'operazione è necessaria quando il task, dopo aver memorizzato alcuni dati nel buffer di traccia, per qualche ragione rileva che non sono più aggiornati. Ovviamente, ETD_CLEAR e CMD_CLEAR non hanno effetto sul contenuto dei buffer definiti dal task.

ETD_FORMAT, TD_FORMAT

Scopo del comando

Questo comando permette a un task di formattare le tracce di un disco, cancellando in maniera irreversibile qualsiasi dato in esse contenuto. Si tratta di un comando altamente flessibile, che permette di formattare anche insiemi ristretti di tracce a partire da qualsiasi posizione sul disco. A seconda del numero di tracce che desidera formattare, il task deve allocare nella chip RAM un buffer da (TD_SECTOR * 11 * N) byte, dove N è il numero di tracce sulle quali s'intende agire. All'interno di questo buffer deve poi memorizzare i dati che verranno trasferiti in blocco nelle tracce su disco a partire da quella indicata. Dal momento che con un elevato numero di tracce da formattare il necessario buffer in memoria diventa molto grande, generalmente si invia un comando ETD_FORMAT, o TD_FORMAT, per ciascuna traccia, di modo che in memoria il task possa allocare un unico buffer da (TD_SECTOR * 11) byte, da impiegare per ogni invio del comando. ETD_FORMAT, come TD_FORMAT, non compie alcuna verifica sulle operazioni che svolge.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si deve inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_FORMAT (o a ETD_FORMAT), io_Flags a 0, e i parametri seguenti ai valori indicati.

- io_Data. Deve contenere l'indirizzo del buffer predisposto dal task nella chip RAM. Questo buffer deve avere le stesse dimensioni delle tracce da formattare, viste come un unico blocco di memoria. Al suo interno il task memorizza il contenuto che le tracce contengono a formattazione ultimata.

- io_Length. Deve contenere la lunghezza, espressa in byte, del buffer puntato da io_Data. Questa lunghezza dev'essere un multiplo di (TD_SECTOR * 11), cioè del numero di byte che compongono una traccia del disco (se non è un multiplo esatto, il comando non ha successo e viene restituito il codice d'errore IOERR_BADLENGTH). Il comando accede a questo parametro per sapere quante tracce deve formattare.

- io_Offset. Deve contenere l'indicazione della traccia dalla quale iniziare la formattazione. Anziché il numero della traccia, però, deve indicare il numero di byte dall'inizio del disco. Se per esempio si desidera che la formattazione inizi dalla traccia 3 compresa (per questo comando le tracce si numerano da 0 a 159), occorre memorizzare in io_Offset il valore (TD_SECTOR * 11 * 3).

- iotd_Count. Se si invia ETD_FORMAT, la versione estesa del comando, occorre memorizzare in questo parametro il numero massimo consentito d'inserimenti e di rimozioni del disco, che il dispositivo confronta con il contatore interno dell'unità per stabilire se eseguire o meno il comando.

Discussione

I comandi ETD_FORMAT e TD_FORMAT vengono utilizzati per formattare le tracce di un disco inserendo al loro interno nuove informazioni. L'operazione di formattazione in genere viene effettuata per organizzare in tracce e settori un nuovo disco, e per rigenerare un disco danneggiato (dal punto di vista logico, non da quello fisico). Al solito, la versione estesa del comando esegue un controllo che tiene conto del valore contenuto nel parametro iotd_Count della richiesta di I/O.

Generalmente i task eseguono la formattazione una traccia alla volta per limitare l'uso di memoria da parte del buffer; inoltre, dopo la formattazione di ogni traccia dovrebbero eseguire una lettura e confrontarne il contenuto con quello del buffer di formattazione per essere certi che non si sono verificati errori. Questa verifica è senza dubbio consigliabile, in quanto né ETD_FORMAT né TD_FORMAT eseguono verifiche sui dati che memorizzano.

Per quanto riguarda i dati che il task memorizza nei parametri io_Length e io_Offset, se non sono multipli esatti di (TD_SECTOR * 11) il comando non viene eseguito e al task viene restituito il codice d'errore IOERR_BADLENGTH.

ETD_MOTOR, TD_MOTOR

Scopo del comando

Questo comando permette a un task di attivare e disattivare la rotazione del motore presente nel disk drive. Il motore del disk drive viene attivato automaticamente durante l'elaborazione della maggior parte dei comandi del dispositivo TrackDisk. Tuttavia non viene disattivato automaticamente: è compito del task utilizzare il comando ETD_MOTOR, o TD_MOTOR, per disattivarlo. Lo stato al quale si desidera portare il motore del disk drive (attivato o disattivato) dev'essere specificato dal task nel parametro io_Length della struttura IOExtTD; un valore pari a 1 avvia la rotazione, mentre 0 la blocca. Lo stato del motore precedente all'esecuzione del comando viene restituito nel parametro io_Actual della struttura di I/O restituita in risposta. Un valore pari a 1 indica che il motore era in rotazione, mentre 0 indica che era fermo. La versione estesa del comando obbliga l'unità a verificare che il disco non sia stato sostituito, prima di procedere all'attivazione o alla disattivazione del motore.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_MOTOR o ETD_MOTOR, io_Flags a 0 e (se s'impiega ETD_MOTOR) iotd_Count con il numero massimo consentito d'inserimenti e di rimozioni del disco.

Discussione

Nella maggior parte dei casi, l'attivazione del motore non risulta necessaria; ogni comando di accesso al disco inviato alle routine interne del dispositivo TrackDisk provvede a farlo autonomamente. Tuttavia, disattivare il motore è compito del task. Si tenga presente che per preservare l'incolumità dei dati si deve togliere il disco dal disk drive solo quando la spia che ne indica l'attività è spenta.

ETD_RAWREAD, TD_RAWREAD

Scopo del comando

Questo comando legge dati grezzi da un'unità del dispositivo TrackDisk. L'unità ricerca la traccia specificata, ne trasferisce il contenuto nel proprio buffer di traccia, e quindi nel buffer del task. I dati che si ottengono vengono definiti "grezzi" in quanto non vengono decodificati dal Blitter. Il task può servirsi del comando esteso se desidera utilizzare il parametro iotd_Count per il controllo degli inserimenti e rimozioni del disco.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a ETD_RAWREAD (o TD_RAWREAD) e i parametri seguenti ai valori indicati.

- io_Flags. Dev'essere inizializzato a 0, oppure a IOTDF_INDEXSYNC se si desidera che il dispositivo TrackDisk legga i byte della traccia a partire dall'identificatore dell'indice presente sulle tracce (index mark). Questa operazione può anche non avere successo. Occorre comunque tener presente che c'è sempre una pausa nel processo di lettura, e può trattarsi anche di una pausa molto lunga se, per esempio, gli interrupt sono stati disabilitati.

- io_Length. Deve contenere la lunghezza del buffer del task espressa in byte; la lunghezza massima è 32K.

- io_Data. Deve contenere un puntatore al buffer del task a cui verranno inviati i dati grezzi prelevati dalla traccia. Questo buffer deve trovarsi nei primi 512K della memoria di sistema (chip RAM).

- io_Offset. Deve contenere il numero della traccia da trasferire nel buffer di traccia. Si noti la differenza rispetto agli altri comandi di accesso al disco per trasferire i dati codificati dal Blitter, come ETD_READ e CMD_READ, i quali si aspettano nel parametro io_Offset il numero di byte dall'inizio del disco dopo i quali deve iniziare la lettura.

- iotd_Count. Se s'invia il comando ETD_RAWREAD anziché TD_RAWREAD, questo parametro rappresenta il massimo numero di estrazioni e d'inserimenti consentito. Durante l'esecuzione del comando questo valore viene confrontato con il contatore interno dell'unità, e se quest'ultimo risulta maggiore il comando non viene eseguito.

- iotd_SecLabel. Dev'essere inizializzato a 0, oppure con un puntatore a una serie di buffer in RAM della lunghezza di 16 byte l'uno; un buffer per ogni settore della traccia in cui devono essere memorizzati i dati riguardanti le etichette durante l'operazione di lettura.

Discussione

ETD_RAWREAD e TD_RAWREAD vengono utilizzati per le operazioni di lettura dal disco dei dati grezzi. Il primo, rispetto al secondo, permette di tener conto delle informazioni riguardanti le sostituzioni del disco, oppure delle informazioni contenute nelle etichette. Con questi comandi non viene effettuata dal coprocessore Blitter alcuna decodifica dei dati contenuti nella traccia; i byte vengono memorizzati nel buffer del task mantenendo lo stesso formato che avevano su disco. Dal momento che i dati sono organizzati nel formato MFM (Modified Frequency Modulation, modulazione di frequenza modificata), conviene ricorrere a questo comando solo nel caso in cui si sappia in che modo vengono definiti e utilizzati i dati nel formato MFM. Inoltre, i programmi che utilizzano questo comando potrebbero non essere compatibili con future versioni del software sistema dell'Amiga.

Se viene impostato il flag IOTDF_INDEXSYNC si verifica sempre una pausa tra la rilevazione dell'impulso indice (index pulse) e l'inizio del trasferimento dei dati dal disco al buffer di traccia dell'unità. Questa pausa è provocata dal tempo che occorre al DMA per prepararsi al trasferimento. La pausa varia da 135 a 200 microsecondi; con 4 microsecondi per bit, ci sono da 4 a 7 byte di pausa; 55 microsecondi vengono utilizzati per la gestione degli interrupt software (è il tempo che trascorre tra l'emissione dell'interrupt e la scrittura nel registro hardware DSKLEN); 64 microsecondi vengono utilizzati per scandire una singola riga di schermo, sincronizzando l'I/O dei dischi con la scansione video effettuata dal chip Agnus. Può essere richiesta una scansione di riga addizionale di 0-65 microsecondi; il registro DSKLEN può essere inizializzato in qualunque punto si trovi il pennello elettronico lungo la scansione orizzontale di una riga di schermo.

ETD_RAWWRITE, TD_RAWWRITE

Scopo del comando

Questo comando trasferisce su disco i byte di dati contenuti nel buffer del task senza codificarli, cioè senza impiegare il Blitter per cambiarne il formato.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a ETD_RAWWRITE (o TD_RAWWRITE) e i parametri seguenti ai valori indicati.

- io_Flags. Dev'essere inizializzato a 0, oppure a IOTDF_INDEXSYNC se si desidera che il dispositivo TrackDisk esegua la scrittura dei dati su disco partendo dall'indice. L'operazione può anche non avere successo. Occorre comunque tenere presente che si verificano sempre delle pause durante il processo di scrittura, che potrebbero anche essere molto lunghe se, per esempio, gli interrupt sono stati disabilitati.

- io_Length. Deve contenere la lunghezza del buffer del task espressa in byte; la lunghezza massima è 32K.

- io_Data. Deve contenere un puntatore al buffer del task da cui i dati vengono trasferiti nella traccia. Questo buffer deve trovarsi nei primi 512K della memoria di sistema (chip RAM).

- io_Offset. Deve contenere il numero della traccia nella quale verranno scritti i dati contenuti nel buffer del task.

- iotd_Count. Se si invia il comando ETD_RAWWRITE, questo parametro rappresenta il massimo numero consentito d'inserimenti e di rimozioni. Durante l'esecuzione del comando questo valore viene confrontato con il contatore interno dell'unità prescelta, e se quest'ultimo risulta maggiore il comando non viene eseguito.

- iotd_SecLabel. Se s'invia il comando ETD_RAWWRITE, si deve inizializzare questo parametro a 0, oppure in modo che punti a una serie

di buffer in RAM della lunghezza di 16 byte l'uno; un buffer per ogni settore al quale il task aggiunge dati riguardanti l'etichetta in modo che anch'essi vengano memorizzati su disco.

Discussione

ETD_RAWWRITE e TD_RAWWRITE vengono utilizzati per le operazioni di scrittura su disco di dati non elaborati. Il primo, a differenza del secondo, permette al task di tener conto delle informazioni riguardanti la sostituzione del disco, oppure delle etichette dei settori. Con questo comando non viene effettuata dal coprocessore Blitter alcuna codifica dei dati contenuti nel buffer del task; i byte vengono memorizzati su disco mantenendo lo stesso formato che avevano nel buffer del task. Considerando che i dati devono essere organizzati nel formato MFM (Modified Frequency Modulation, modulazione di frequenza modificata), conviene utilizzare questo comando soltanto se si ha una completa padronanza del formato MFM. Inoltre, i programmi che fanno uso di questo comando potrebbero non essere compatibili con future versioni del software sistema dell'Amiga.

ETD_READ, CMD_READ

Scopo del comando

Questo comando legge dati da un'unità del dispositivo TrackDisk facendoli codificare dal Blitter. L'operazione di decodifica assicura che i byte ricevuti nel buffer del task sono proprio quelli che erano stati salvati su disco. Il task indica il settore dal quale deve iniziare la lettura e il numero di settori da leggere. Quando il comando viene eseguito, il dispositivo individua la traccia contenente il settore indicato dal task, trasferisce il contenuto della traccia nel suo buffer di lettura interno e infine trasferisce i settori che il task ha indicato nel relativo buffer di lettura. Il task deve servirsi del comando esteso se desidera avere il controllo del parametro iotd_Count.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati

dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a ETD_READ (o CMD_READ) e i parametri seguenti ai valori indicati.

- io_Flags. Dev'essere inizializzato a 0.

- io_Length. Deve contenere la lunghezza del buffer del task espressa in byte. Questo numero dev'essere un multiplo della capienza di un settore (TD_SECTOR, pari a 512 byte). Da questo parametro si deduce ovviamente quanti sono i settori da leggere.

- io_Data. Deve contenere un puntatore al buffer del task nel quale verranno memorizzati i byte contenuti nei settori del disco che il task ha indicato; i byte vengono codificati dal Blitter, e per questa ragione il buffer deve trovarsi nella chip RAM.

- io_Offset. Deve contenere il numero di byte dopo i quali avrà inizio la lettura (contati dall'inizio del disco). Questo numero dev'essere un multiplo di TD_SECTOR, e può essere facilmente calcolato tramite la seguente formula:

```
iOExtTD->io_Offset = TD_SECTOR * (settore + NUMSECS * faccia +
                     NUMSECS * traccia * NUMHEADS);
```

  all'interno della quale TD_SECTOR (512), NUMSECS (11) e NUMHEADS (2) sono costanti definite nel file INCLUDE trackdisk.h; "settore" può variare da 0 a 10, "faccia" da 0 a 1 e "traccia" da 0 a 79. Se per esempio desiderassimo accedere al settore 0 della traccia 1 sulla faccia 1, dovremmo memorizzare il valore 16.896. Per comprendere il funzionamento di questa formula si tenga presente che se nel parametro io_Offset viene memorizzato un numero multiplo di 512 compreso fra 0 e 5.120, CMD_READ accede alla traccia 0 della faccia 0; se in io_Offset viene memorizzato un numero multiplo di 512 compreso fra 5.632 e 10.752, CMD_READ accede alla traccia 0 della faccia 1 (si noti che quindi con una lettura sequenziale della traccia 0, prima sulla faccia 0 e poi sulla faccia 1, la testina di lettura non si muove); se in io_Offset viene memorizzato un numero compreso fra 11.264 e 16.384, CMD_READ accede alla traccia 1 della faccia 0, e così via.

- iotd_Count. Se si invia il comando ETD_READ anziché CMD_READ, questo parametro rappresenta il massimo numero consentito di estrazioni e d'inserimenti. Durante l'esecuzione del comando questo valore viene confrontato con quello riportato nel contatore interno dell'unità, e se quest'ultimo risulta maggiore il comando non viene eseguito.

Discussione

ETD_READ e CMD_READ vengono utilizzati per le operazioni di lettura dal disco di settori di byte codificati. Il primo, a differenza del secondo, permette al task di tener conto delle informazioni riguardanti la sostituzione del disco. Con questi comandi durante il trasferimento il Blitter decodifica i byte grezzi prima che giungano nel buffer del task.

ETD_SEEK, TD_SEEK

Scopo del comando

Questo comando permette a un task di muovere la testina del disk drive, controllando così la posizione sulla quale avverrà il prossimo accesso in lettura o in scrittura. Il parametro io_Offset della struttura IOExtTD deve indicare la traccia su cui si desidera collocare la testina. Questo comando non legge alcun dato da disco. Il task deve servirsi del comando esteso se desidera avere il controllo del parametro iotd_Count.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_SEEK (o ETD_SEEK), io_Flags a 0 e io_Offset con il numero della traccia desiderata espresso in byte dall'inizio del disco (le tracce sono numerate da 0 a 159, e quindi data la traccia N, il valore da memorizzare nel parametro io_Offset è TD_SECTOR * 11 * N). Se s'impiega la versione estesa del comando occorre anche memorizzare il numero massimo consentito d'inserimenti e di rimozioni del disco nel parametro iotd_Count.

Discussione

La testina del drive viene mossa automaticamente sull'appropriata traccia durante l'esecuzione dei comandi del dispositivo TrackDisk; quindi in genere non è necessario impartire esplicitamente alcun comando ETD_SEEK, o

TD_SEEK. Tuttavia con questo comando un task può provocare lo spostamento della testina collocandola su una specifica traccia del disco. Questa capacità di "preposizionamento" aumenta la velocità d'accesso al disco sia in lettura sia in scrittura. La versione estesa del comando effettua un controllo che tiene conto del valore contenuto nel parametro iotd_Count.

ETD_UPDATE, CMD_UPDATE

Scopo del comando

Questo comando permette a un task di ordinare a un'unità il trasferimento su disco del contenuto del buffer di traccia. Il contenuto del buffer del task non viene influenzato da questa operazione.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a ETD_UPDATE (o CMD_UPDATE), io_Flags a 0 e iotd_Count con il numero di inserimenti e rimozioni consentito (soltanto se s'invia il comando esteso).

Discussione

I comandi ETD_UPDATE e CMD_UPDATE permettono a un task di ordinare a un'unità l'immediato trasferimento del suo buffer di traccia su disco. Quest'operazione viene effettuata automaticamente nel caso di una caduta di tensione oppure di un reset impartito dall'utente, ma un task può voler forzare l'accesso in scrittura sul disco per altre ragioni. L'operazione di scrittura automatica avviene solo se il contenuto del buffer è stato modificato dopo l'operazione di lettura. Questo comando non influenza il contenuto del buffer definito dal task. Ricordiamo che se il contenuto del parametro iotd_Count risulta inferiore al valore presente nel contatore interno dell'unità, il comando ETD_UPDATE non viene eseguito.

ETD_WRITE, CMD_WRITE

Scopo del comando

Questo comando memorizza un blocco di dati sul disco che si trova nel disk drive selezionato; il dispositivo, prima di salvare i dati su disco, provvede a codificarli tramite il Blitter. Il task indica il settore del disco nel quale deve iniziare la scrittura e il numero di settori da scrivere. Il dispositivo individua la traccia contenente il settore indicato dal task, trasferisce il contenuto del buffer di scrittura del task nel suo buffer di scrittura interno e infine trasferisce l'intero buffer interno su disco. Il task deve servirsi del comando esteso se desidera avere il controllo del parametro iotd_Count.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a ETD_WRITE (o CMD_WRITE) e i parametri seguenti ai valori indicati.

- io_Flags. Dev'essere inizializzato a 0.
- io_Length. Deve contenere la lunghezza del buffer del task espressa in byte. Questo numero dev'essere un multiplo della capienza di un settore (TD_SECTOR, pari a 512 byte). Da questo parametro si deduce ovviamente quanti sono i settori da salvare su disco.
- io_Data. Deve contenere un puntatore al buffer nel quale il task ha memorizzato i byte da trasferire sul disco; i byte vengono codificati dal Blitter, e per questa ragione il buffer deve trovarsi nella chip RAM.
- io_Offset. Indica la distanza in byte tra l'inizio del disco e il punto in cui deve iniziare la scrittura. Questo numero dev'essere un multiplo di TD_SECTOR, e può essere facilmente calcolato tramite la seguente formula:

```
iOExtTD->io_Offset = TD_SECTOR * (settore + NUMSECS * faccia +
                     NUMSECS * traccia * NUMHEADS);
```

all'interno della quale TD_SECTOR (512), NUMSECS (11) e NUMHEADS (2) sono costanti definite nel file INCLUDE trackdisk.h: "settore" può variare da 0 a 10, "faccia" da 0 a 1 e "traccia" da 0 a 79. Se per esempio desiderassimo accedere al settore 0 della traccia 1 sulla faccia 1, dovremmo memorizzare il valore 16.896. Per comprendere il funzionamento di questa formula si tenga presente che se nel parametro io_Offset viene memorizzato un numero multiplo di 512 compreso fra 0 e 5.120, CMD_WRITE accede alla traccia 0 della faccia 0; se in io_Offset viene memorizzato un numero multiplo di 512 compreso fra 5.632 e 10.752, CMD_WRITE accede alla traccia 0 della faccia 1 (si noti che quindi con una scrittura sequenziale della traccia 0 della faccia 0, e della traccia 0 della faccia 1, la testina non si muove); se in io_Offset viene memorizzato un numero compreso fra 11.264 e 16.384, CMD_WRITE accede alla traccia 1 della faccia 0, e così via.

- iotd_Count. Se s'invia il comando ETD_WRITE anziché CMD_WRITE, si deve inizializzare questo parametro con il massimo numero consentito di estrazioni e inserimenti. Durante l'esecuzione del comando questo valore viene confrontato con il contatore interno dell'unità, e se quest'ultimo risulta maggiore il comando non viene eseguito.

Discussione

ETD_WRITE e CMD_WRITE vengono utilizzati per la scrittura su disco di byte codificati. Il primo, a differenza del secondo, permette di tener conto delle informazioni riguardanti eventuali sostituzioni del disco. Con questi comandi Blitter codifica i byte grezzi prima che vengano effettivamente memorizzati su disco.

TD_ADDCHANGEINT

Scopo del comando

Questo comando aggiunge al software sistema un nuovo meccanismo di gestione degli interrupt causati dalla rimozione e dall'inserimento dei dischi. È stato realizzato perché il comando TD_REMOVE non risultava sufficientemente veloce nella gestione delle funzioni di interrupt aggiuntive.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_ADDCHANGEINT, io_Flags a 0, e io_Data con l'indirizzo di una struttura Interrupt che rappresenta la nuova routine di gestione degli interrupt.

Discussione

Il dispositivo TrackDisk mantiene una lista di funzioni di gestione degli interrupt software generati dall'inserimento e dalla rimozione dei dischi dai disk drive. Il comando TD_ADDCHANGEINT permette ai task di aggiungere qualsiasi routine di gestione degli interrupt personalizzata a questa lista. Si noti che la richiesta non viene restituita fino a quando il task non invia il comando TD_REMCHANGEINT per rimuovere la routine.

TD_CHANGENUM

Scopo del comando

Questa funzione restituisce nel parametro io_Actual della struttura IOExtTD un numero che rappresenta la quantità di rimozioni e d'inserimenti di dischi avvenuti nell'unità prescelta.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si deve inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_CHANGENUM e io_Flags a 0.

Discussione

Il contatore interno di ogni unità è molto importante nello svolgimento delle operazioni che richiedono l'impiego dei comandi estesi del dispositivo TrackDisk. Ogni volta che un disco viene inserito o rimosso da un particolare disk drive, il relativo contatore interno viene incrementato di 1. Quando durante un accesso in lettura o in scrittura richiesto tramite un comando esteso il valore contenuto nel contatore risulta maggiore del valore indicato dal task nel parametro iotd_Count della struttura IOExtTD, il comando esteso non effettua l'accesso e restituisce il codice d'errore TDERR_DiskChanged. Questo controllo evita che vecchie richieste di I/O agiscano sul disco presente nel disk drive quando non si tratta del disco desiderato. Inoltre, il controllo eseguito dai comandi estesi evita l'impiego del disco sbagliato dopo una serie di sostituzioni.

Se un task vuole utilizzare i comandi estesi ma non desidera controlli sul contatore interno dell'unità, deve impostare il parametro iotd_Count con il massimo valore possibile (0xFFFFFFFF).

TD_CHANGESTATE

Scopo del comando

Questo comando rileva se è presente un disco nell'unità specificata. Se nel disk drive non è presente alcun disco, nel parametro io_Actual della struttura IOExtTD viene restituito un valore diverso da 0.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si deve inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_CHANGESTATE e io_Flags a 0.

Discussione

TD_CHANGESTATE permette a un task di rilevare se in un disk drive è presente o meno un disco. Il task deve richiedere quest'informazione prima di qualsiasi accesso in lettura o scrittura a un'unità del dispositivo.

TD_GETDRIVETYPE

Scopo del comando

Questo comando restituisce il tipo di disk drive corrispondente all'unità indirizzata. Le costanti che rappresentano i vari tipi di disk drive sono numeri interi a 8 bit (short integer), e sono definiti come costanti nei file INCLUDE trackdisk.h e trackdisk.i. Il tipo di disk drive viene restituito nel parametro io_Actual della struttura IOStdReq appartenente alla struttura IOExtTD. Il valore 1 (DRIVE3_5) indica un disk drive per dischi da 3,5", mentre il valore 2 (DRIVE5_25) indica un disk drive per dischi da 5,25".

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_GETDRIVETYPE e io_Flags a 0.

Discussione

Una chiamata alla funzione OpenDevice effettuata da un task non ha successo se le routine interne del dispositivo non riconoscono un particolare disk drive associato all'unità del dispositivo TrackDisk indicata nella chiamata.

TD_GETNUMTRACKS

Scopo del comando

Con questo comando si ottiene nel parametro io_Actual della struttura IOExtTD il numero di tracce disponibili nell'unità indicata. Il numero viene calcolato dalle routine interne del dispositivo TrackDisk basandosi sulle caratteristiche fisiche del disk drive collegato.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata alla funzione OpenDevice. Si devono inizializzare infine io_Command a TD_GETNUMTRACKS e io_Flags a 0.

Discussione

Il comando TD_GETNUMTRACKS rimuove alcune restrizioni presenti nella release 1.1 (e precedenti) del software sistema. In particolare, questo comando rende inutile la costante NUMTRACKS contenuta nel file INCLUDE trackdisk.h delle versioni 1.0 e 1.1, alla quale non si dovrebbe quindi mai fare riferimento.

TD_PROTSTATUS

Scopo del comando

Questo comando permette a un task di verificare lo stato della linguetta di protezione di un disco. Lo stato della protezione viene restituito nel parametro io_Actual della struttura IOExtTD. Il valore 0 indica che il disco non è protetto

(linguetta chiusa) mentre un valore diverso da 0 indica che il disco è protetto (linguetta aperta). Se non è presente alcun disco nel disk drive, il parametro io_Error della struttura IOExtTD viene restituito con il codice d'errore TDERR_DiskChanged. Si tenga presente che i disk drive dispongono di un congegno hardware adibito a interdire ogni accesso in scrittura quando la linguetta di protezione del disco inserito è aperta. Questo congegno hardware non è governabile via software e quindi garantisce la massima sicurezza dei dati su disco quando la linguetta è aperta.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_PROTSTATUS e io_Flags a 0.

Discussione

TD_PROTSTATUS permette a un task di rilevare lo stato della linguetta di protezione del disco presente nel disk drive associato all'unità. Il task deve sempre sapere se l'accesso in scrittura è consentito, prima di procedere alla scrittura dei dati su disco (la lettura, invece, può avvenire in qualsiasi caso).

TD_REMCHANGEINT

Scopo del comando

Questo comando rimuove dal sistema la funzione di interrupt per la gestione della sostituzione del disco che il task attiva tramite il comando TD_ADDCHANGEINT. Al momento dell'invio di TD_REMCHANGEINT viene restituita la richiesta di I/O che era stata utilizzata per inviare il comando TD_ADDCHANGEINT.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_REMCHANGEINT, io_Flags a 0 e io_Data in modo che punti alla struttura Interrupt relativa alla funzione da rimuovere.

Discussione

Il dispositivo TrackDisk mantiene una lista di funzioni di gestione degli interrupt software generati dall'inserimento e dalla rimozione dei dischi da un disk drive. I comandi TD_ADDCHANGEINT e TD_REMCHANGEINT permettono ai task di aggiungere e rimuovere qualsiasi funzione personalizzata di interrupt relativa alla rimozione di un disco.

TD_REMOVE

Scopo del comando

Questo comando permette a un task di aggiungere una funzione di interrupt per la gestione delle sostituzioni di dischi, tramite la quale effettuare determinate azioni quando un utente inserisce o rimuove un disco dal disk drive.

Preparazione della struttura IOExtTD

Si deve inizializzare mn_ReplyPort in modo che punti alla struttura MsgPort che rappresenta la reply port del task. Si devono inoltre inizializzare io_Device e io_Unit in modo che puntino rispettivamente alle strutture Device e Unit che gestiscono l'unità indirizzata; questi parametri devono essere copiati dalla struttura IOExtTD inizializzata con la prima chiamata a OpenDevice. Si devono inizializzare infine io_Command a TD_REMOVE, io_Flags a 0 e io_Data in modo che punti alla struttura Interrupt che descrive la funzione da aggiungere.

Discussione

TD_REMOVE serve per far sì che venga eseguita una specifica funzione di interrupt ogni volta che viene sostituito il disco nel disk drive. La struttura Interrupt passata come parametro deve contenere un puntatore ai dati e ai codici che vengono utilizzati per gestire l'interrupt prodotto ogni volta che un disco viene rimosso o inserito. Se io_Data risulta nullo, gli interrupt di gestione della sostituzione del disco vengono sospesi.

Appendice

Definizioni in C delle funzioni di supporto alla libreria Exec

Questa appendice presenta le definizioni in linguaggio C delle funzioni di supporto alla libreria Exec discusse nel capitolo 2. La funzione NewList non è stata inclusa in questa trattazione, perché agisce chiamando una macro in linguaggio Assembly.

Le funzioni di supporto alla libreria Exec vengono utilizzate dai task perlopiù per gestire i dispositivi. Tuttavia, sono utilizzabili in qualsiasi programma che si serva di task, message port, strutture per le richieste di I/O (standard o estese) e liste. In questa appendice ne mostriamo i listati in linguaggio C al fine di renderne evidente il funzionamento e di offrire al programmatore un riferimento nella creazione dei suoi programmi.

Le funzioni di supporto alla libreria Exec si trovano nel file amiga.lib per il compilatore C della Lattice, nel file c.lib per il compilatore Aztec C68k della Manx (si tratta ovviamente di due librerie linked, cioè di librerie che vengono chiamate durante la fase di link per risolvere i riferimenti alle funzioni esterne, in contrapposizione con le librerie shared dell'Amiga che vengono invece aperte durante l'esecuzione dei task). Per chiamarle all'interno di un programma è quindi sufficiente dichiarare che tipo di risultati restituiscono, in modo tale che il compilatore, pur senza conoscerle, sappia almeno se c'è compatibilità di tipo con le variabili nelle quali vengono memorizzati i risultati. Per fare un esempio, se all'interno di un generico programma impieghiamo le funzioni CreatePort e CreateStdIO, all'esterno della funzione principale main() dovrebbero trovarsi le seguenti dichiarazioni:

```
struct MsgPort *CreatePort();
struct IOStdReq *CreateStdIO();
```

Nel pacchetto della Manx, all'interno della directory include, esiste un file di nome functions.h nel quale sono definiti i tipi di dati restituiti da tutte le funzioni previste dall'Amiga. Se nel sorgente viene incluso anche questo file, le suddette dichiarazioni sono superflue, eccetto che per le funzioni CreateExtIO e DeleteExtIO, che non appaiono nel file functions.h. Comunque, per ragioni di compatibilità fra i vari compilatori è sempre opportuno effettuare le citate dichiarazioni.

Le funzioni di supporto alla libreria Exec servono ad allocare e soprattutto a inizializzare particolari strutture di uso frequente, evitando ai programmatori inutili fatiche ed errori.

CreateExtIO

Il listato che segue mostra le istruzioni in linguaggio C utilizzate per definire la funzione CreateExtIO(). Esso chiarisce le dimensioni degli argomenti che devono essere passati nella chiamata; un errore diffuso è quello di passare come argomento size un valore da 16 bit anziché 32, errore che generalmente si verifica quando il programma viene compilato in modo che gli

interi siano da 16 bit. Osservando il listato si può comprendere anche in che modo vengono inizializzati i parametri della struttura IORequest e come viene allocata la memoria per la struttura grazie alla funzione AllocMem. Inoltre viene mostrato come si calcola la disponibilità di memoria, come si cambia in C il tipo delle variabili coinvolte (operazione di cast) e come si specificano i parametri in una struttura che possiede sotto-strutture nidificate (la sotto-struttura Message della struttura IORequest possiede a sua volta una sotto-struttura Node). Si noti infine che la funzione memorizza l'estensione della struttura estesa non standard nel parametro mn_Length della sotto-struttura Message, in modo che la corrispondente funzione DeleteExtIO possa sapere quanti byte deve disallocare. Il parametro mn_Length della struttura Message, normalmente impiegato per indicare la lunghezza del messaggio che segue l'intestazione, non viene infatti impiegato dai dispositivi.

```
struct IORequest *CreateExtIO (iOReplyPort, size)
struct MsgPort *iOReplyPort;
ULONG size;
{
     struct IORequest *myIOExtReq;
     if (iOReplyPort == ØL) return ((struct IORequest *) ØL);
     myIOExtReq = (struct IORequest *) AllocMem (size, MEMF_CLEAR |
                  MEMF_PUBLIC);
     if (myIOExtReq == ØL) return ((struct IORequest *) ØL);
     myIOExtReq->io_Message.mn_Node.ln_Type = NT_MESSAGE;
     myIOExtReq->io_Message.mn_Node.ln_Pri = Ø;
     myIOExtReq->io_Message.mn_Length = (UWORD)size;
     myIOExtReq->io_Message.mn_ReplyPort = iOReplyPort;
     return (myIOExtReq);
}
```

CreatePort

Il listato che segue mostra le istruzioni in linguaggio C necessarie alla definizione della funzione CreatePort. Chiarisce le dimensioni degli argomenti che le devono essere passati nella chiamata, il modo in cui vengono inizializzati i parametri della struttura MsgPort, e come viene allocata la memoria per la struttura utilizzando la funzione AllocMem. Viene mostrato come si rileva la disponibilità di memoria, come si cambia in C il tipo delle variabili coinvolte (operazione di cast), come si assegna a una message port un bit di segnale del task e come lo si libera, come si individua l'indirizzo della struttura Task che definisce il task. Inoltre, il listato evidenzia come si specificano i parametri in una struttura che possiede sotto-strutture nidificate (la struttura MsgPort possiede una sotto-struttura Node), come si aggiungono le message port alla lista di sistema delle message port pubbliche, e come si crea una nuova lista.

```
struct MsgPort *CreatePort (msgPortName, msgPort_Priority)
char *msgPortName;
LONG msgPort_Priority;
{
     UBYTE msgPort_SignalBitNumber;
     struct MsgPort *myMsgPort;
     if ((msgPort_SignalBitNumber = AllocSignal (-1L)) == -1L)
          return ((struct MsgPort *) 0L);
     myMsgPort = (struct MsgPort *) AllocMem ((LONG) sizeof(struct MsgPort),
                    MEMF_CLEAR | MEMF_PUBLIC);
     if (myMsgPort == 0L)
     {
          FreeSignal (msgPort_SignalBitNumber);
          return ((struct MsgPort *) 0L);
     }
     myMsgPort->mp_Node.ln_Name = msgPortName;
     myMsgPort->mp_Node.ln_Pri = (BYTE)msgPort_Priority;
     myMsgPort->mp_Node.ln_Type = NT_MSGPORT;
     myMsgPort->mp_Flags = PA_SIGNAL;
     myMsgPort->mp_SigBit = msgPort_SignalBitNumber;
     myMsgPort->mp_SigTask = (struct Task *)FindTask (0L);
     if (msgPortName != 0L)
          AddPort (myMsgPort);
     else
          NewList (&(myMsgPort->mp_MsgList));
     return (myMsgPort);
}
```

CreateStdIO

Il listato che segue mostra le istruzioni in linguaggio C necessarie alla definizione della funzione CreateStdIO. Chiarisce le dimensioni degli argomenti che le devono essere passati nella chiamata, il modo in cui vengono inizializzati i parametri della struttura IOStdReq e come viene allocata la memoria per la struttura utilizzando la funzione AllocMem. Inoltre viene mostrato come si rileva la disponibilità di memoria, come si cambia in C il tipo delle variabili coinvolte (operazione di cast), come si specificano i parametri in una struttura che possiede sotto-strutture nidificate (la struttura IOStdReq possiede una sotto-struttura Message che a sua volta possiede una sotto-struttura Node).

```
struct IOStdReq *CreateStdIO (iOReplyPort)
struct MsgPort *iOReplyPort;
{
     struct IOStdReq *myIOStdReq;
```

```
    if (iOReplyPort == ØL) return ((struct IOStdReq *) ØL);
    myIOStdReq = (struct IOStdReq *) AllocMem ((LONG)sizeof(struct IOStdReq),
                 MEMF_CLEAR | MEMF_PUBLIC);
    if (myIOStdReq == ØL) return ((struct IOStdReq *) ØL);
    myIOStdReq->io_Message.mn_Node.ln_Type = NT_MESSAGE;
    myIOStdReq->io_Message.mn_Node.ln_Pri = Ø;
    myIOStdReq->io_Message.mn_ReplyPort = iOReplyPort;
    return (myIOStdReq);
}
```

CreateTask

Il listato che segue mostra le istruzioni in linguaggio C necessarie alla definizione della funzione CreateTask. Chiarisce le dimensioni degli argomenti che le devono essere passati nella chiamata, il modo in cui vengono inizializzati i parametri della struttura Task e come viene allocata la memoria per la struttura utilizzando la funzione AllocMem. Inoltre viene mostrato come si calcola la disponibilità di memoria, come si alloca uno stack per il task, come si cambia in C il tipo delle variabili coinvolte (operazione di cast), il modo in cui si assegnano le variabili in linguaggio C, come si specificano i parametri in una struttura che possiede sotto-strutture nidificate (la struttura IOStdReq possiede una sotto-struttura Message che a sua volta possiede una sotto-struttura Node). Infine, mostra il modo in cui tramite la funzione AddTask si può aggiungere un task alla lista di sistema.

```
struct Task *CreateTask (myTaskName, myTaskPriority,
                         task_EntryPoint, task_StackSize)
char *myTaskName;
UBYTE myTaskPriority;
APTR task_EntryPoint;
ULONG task_StackSize;
{
    struct Task *myTask;
    ULONG dataSize = (task_StackSize & ØxFFFFFC) + 4;
    myTask = (struct Task *) AllocMem ((LONG)sizeof(struct Task) + dataSize,
             MEMF_CLEAR | MEMF_PUBLIC);
    if (myTask == ØL) return ((struct Task *) ØL));
    myTask->tc_SPLower = (APTR)(myTask + (LONG) sizeof(struct Task));
    myTask->tc_SPUpper = (APTR)((ULONG)(myTask->
                                        tc_SPLower + dataSize) & ØxFFFFFC);
    myTask->tc_SPReg = myTask->tc_SPUpper;
    myTask->tc_Node.ln_Type = NT_TASK;
    myTask->tc_Node.ln_Pri = (BYTE)myTaskPriority;
    myTask->tc_Node.ln_Name = myTaskName;
    AddTask (myTask, task_EntryPoint, ØL);
```

```
    return (myTask);
}
```

DeleteExtIO

Il listato che segue mostra le istruzioni in linguaggio C necessarie alla definizione della funzione DeleteExtIO. Chiarisce le dimensioni dell'argomento che le deve essere passato nella chiamata, il modo in cui vengono cambiati i parametri in una struttura e come viene liberata la memoria occupata dalla struttura di I/O estesa non standard. Si noti che la funzione si affida al contenuto del parametro mn_Length della sotto-struttura Message per rilevare quanti byte di memoria deve liberare (in questo parametro si aspetta di trovare l'opportuno valore memorizzato dalla funzione CreateExtIO). Se per creare la struttura di I/O estesa non standard il task non ha impiegato la funzione CreateExtIO, ma desidera ugualmente impiegare DeleteExtIO per rimuoverla, deve obbligatoriamente memorizzare nel parametro mn_Length la sua estensione in byte. Si noti infine che per avere la certezza che la struttura non verrà più riutilizzata, la funzione oltre a disallocarla provvede a impostarne con valori impropri alcuni parametri.

```
void DeleteExtIO (myIOExtReq)
struct IORequest *myIOExtReq;
{
    myIOExtReq->io_Message.mn_Node.ln_Type = 0xFF;
    myIOExtReq->io_Device = (struct Device *) -1L;
    myIOExtReq->io_Unit = (struct Unit *) -1L;
    FreeMem (myIOExtReq, (ULONG) myIOExtReq->io_Message.mn_Length);
}
```

DeletePort

Il listato che segue mostra le istruzioni in linguaggio C necessarie a definire la funzione DeletePort. Chiarisce la dimensione dell'argomento che occorre passarle nella chiamata, come viene rimossa una message port dalla lista di sistema delle message port pubbliche, come viene liberato il bit di segnale a essa assegnato e la memoria da essa occupata. Si noti infine che per avere la certezza che la struttura non verrà più riutilizzata, la funzione oltre a disallocarla provvede a impostarne alcuni parametri con valori impropri.

```
void DeletePort (myMsgPort)
struct MsgPort *myMsgPort;
```

```
{
    if ((myMsgPort->mp_Node.ln_Name) != ØL)
        RemPort (myMsgPort);
    myMsgPort->mp_Node.ln_Type = ØxFF;
    myMsgPort->mp_MsgList.lh_Head = (struct Node *) -1L;
    FreeSignal (myMsgPort->mp_SigBit);
    FreeMem (myMsgPort, (ULONG)sizeof(struct MsgPort));
}
```

DeleteStdIO

Il listato che segue mostra le istruzioni in linguaggio C necessarie a definire la funzione DeleteStdIO. Chiarisce la dimensione dell'argomento che occorre passarle nella chiamata, il modo in cui vengono cambiati i parametri in una struttura, e come viene liberata la memoria assegnata a quella struttura. Si noti infine che per avere la certezza che la struttura non verrà più riutilizzata, la funzione oltre a disallocarla provvede a impostarne alcuni parametri con valori impropri.

```
void DeleteStdIO (myIOStdReq)
struct IOStdReq *myIOStdReq;
{
    myIOStdReq->io_Message.mn_Node.ln_Type = ØxFF;
    myIOStdReq->io_Device = (struct Device *) -1L;
    myIOStdReq->io_Unit = (struct Unit *) -1L;
    FreeMem (myIOStdReq, (ULONG)sizeof(struct IOStdReq));
}
```

DeleteTask

Il listato che segue mostra le istruzioni in linguaggio C necessarie per definire la funzione DeleteTask. Chiarisce la dimensione dell'argomento che occorre passarle nella chiamata, come viene rimosso un task dalla lista di sistema dei task, e come viene liberata la memoria assegnata alla struttura Task.

```
void DeleteTask (myTask)
struct Task *myTask;
{
    RemTask (myTask);
    FreeMem (myTask, 1 + (LONG) (myTask->tc_SPUpper) - (LONG)myTask);
}
```

Indice analitico

A

AbortIO, funzione, 57
Accesso ai dispositivi, XXX
  condiviso ed esclusivo, 13
ADCMD_ALLOCATE, comando, 84-86, 88, 90-93, 95, 101, 105, 107, 123, 128, 131, 134
ADCMD_FINISH, comando, 91, 97, 110, 112, 119-120, 126-127
ADCMD_FREE, comando, 86, 89, 92-93, 98, 100, 112-113, 128, 131
ADCMD_LOCK, comando, 86-88, 91, 98, 100-102, 113, 119, 125-126, 128-129
ADCMD_PERVOL, comando, 91, 96-97, 112, 120-123, 131
ADCMD_SETPREC, comando, 87, 125, 130, 133
ADCMD_WAITCYCLE, comando, 91, 109, 112, 134
AddDevice, funzione, 2, 63
AddTask, funzione, 39
AddTime, funzione, 415, 422, 424, 427-428, 430
Altoparlanti, assegnazione ai canali audio, 79-80
Amiga, computer, XXIII
  ambiente di programmazione previsto dal, XXXI-XXXV
  capacità del, XXVI
  versioni del sistema operativo del, 282, 456, 458
AmigaDOS, 238, 305, 316, 329
  funzionamento del dispositivo Keyboard e, 305
ANSI X3.64, standard, 269-270, 286-288, 290-293, 295, 351-355, 361, 372-374, 377-378, 380
Array dei caratteri di EOF
  del dispositivo Parallel, 176, 180, 195
  del dispositivo Serial, 205, 212, 231
Array della forma d'onda, 79-82
Array di combinazioni dei canali audio, 82
ASCII, caratteri, 177, 206, 267, 270, 293
  finestre di Intuition e, 269
Asincrono, richieste di I/O, 41
Assegnazione dei canali audio, 82, 87, 123
Assembly, linguaggio. *Vedere* C, linguaggio
Attenuazione del suono, 80
AudChannel, struttura, 98-100
Audio, comandi del dispositivo per, 90-93, 108-136
  assegnare una combinazione di canali audio, 87, 123
  attendere la fine del ciclo della forma d'onda, 134
  azzerare i buffer interni, 108
  bloccare i canali audio, 116
  eliminare richieste di I/O, 109
  eliminare una richiesta di I/O, 122-123, 127
  leggere da un canale, 111
  liberare e assegnare canali audio, 82, 87, 123, 128
  modificare il volume e il periodo, 131
  modificare la priorità d'assegnazione, 133
  produrre un suono, 119
  proteggere canali audio, 85, 129
  riattivare i canali audio, 114
  trasferire il contenuto dei buffer all'hardware, 118
Audio, dispositivo, 79-136
  accesso del dispositivo Narrator al, 144
  assegnazione dei canali audio, 82, 87, 123
  BeginIO e, 79
  canali audio 79
  chiave d'assegnazione, 87, 95
  codici d'errore, 101
  comandi specifici del, 123-136
  comandi standard del, 108-123
  invio dei comandi del, 90
  priorità d'assegnazione dei canali audio, 85
  sistema hardware del, 79-82
  stringa di fonemi, 139
  strutture di I/O del, 31-32, 93
Audio, funzioni del dispositivo, 108-134
Audio, strutture del dispositivo, 93

B

BeginIO, funzione, 4, 7, 40-42, 45-47, 49-50, 58
Bit
  di segnale, 44
  matrice di tastiera, 318
  porta parallela, 174-176
  porta parallela, registro di stato della, 173
  porta seriale, registro di stato della, 200, 202
  vedere anche Flag
BLink, linker, XXXIV
Blitter, coprocessore, 438, 455, 457
Bocca, sagoma della, 139, 146, 159, 163
  parametri per la, 146
  Voice, sotto-struttura, 146
BootBlock, struttura, 447
Break, segnale di, 202, 207, 228
Buffer, XXVII
  comandi di gestione dei, interni dei dispositivi, 19
  dispositivo Parallel, del, 171
  double-buffered, modo, 79, 119
  interni dei dispositivi, 4
  interno del dispositivo Serial, 199
Buffer, definiti dal task, XXVII, 4, 189, 192
  con più message port, 10
  operazioni di lettura e scrittura con i, 171

C

C, linguaggio,
  Exec, definizione delle funzioni di supporto alla libreria, 473-478
  Lattice, compilatore C, XXXI
  Manx, compilatore C, XXXI, 37
  procedure di programmazione in, 37
C, subdirectory, XXXIII
Cache, 20
Caduta di tensione, 22, 461
Campionamento del suono, periodo di, 79-82
Canali audio, array di combinazioni dei, 82
Canali audio, assegnazione dei, 82, 87, 123
Canali audio, priorità d'assegnazione dei, 85, 144
Canali DMA, 79
Caratteri
  ANSI X3.64, standard, 269-270, 286-288, 290-293, 295, 351-355, 361, 372-374, 377-378, 380

di controllo dello schermo, 293
CBD_CURRENTREADID, comando, 389, 407
CBD_CURRENTWRITEID, comando, 388-389, 408
CBD_POST, comando, 387-389, 393, 395-396, 407-409, 410
CD_ASKDEFAULTKEYMAP, comando, 296
CD_ASKKEYMAP, comando, 297
CD_SETDEFAULTKEYMAP, comando, 298
CD_SETKEYMAP, comando, 299
CDInputHandler, funzione, 281
Centronics, interfaccia parallela, 172
CheckIO, funzione, 4, 7, 44, 62
Chiave d'assegnazione, 87, 95
Chip RAM, XXVI, 79, 96
CIA, Complex Interface Adapter, 417
Cicli di clock, 81, 96
CLI, Command Line Interface, 351
Clipboard, comandi del dispositivo per, 392, 400-411
effettuare il reset dell'unità, 402
finire la scrittura di un clip, 403
leggere un clip, 400
ottenere l'identificatore di lettura, 407
ottenere l'identificatore di scrittura, 408
prorogare un clip, 410
salvare un clip, 404
Clipboard, dispositivo, 385-411
clip privati e pubblici del, 385-386
comandi specifici del, 407
comandi standard del, 400
cut & paste con il, 385-386
funzionamento del, 387
operazione sequenziali di lettura e scrittura del, 389
strutture di I/O del, 393
Clipboard, file, 385
Clipboard, funzioni del dispositivo, 397
Clipboard, strutture del dispositivo, 393
ClipboardUnitPartial, struttura, 394
CLIPS:, directory logica, 386
Clock, frequenza di, 333
CloseDevice, funzione, XXVII, 20
dispositivo Audio, 103
dispositivo Clipboard, 397
dispositivo Console, 283
dispositivo Gameport, 334
dispositivo Input, 247
dispositivo Keyboard, 308
dispositivo Narrator, 149
dispositivo Parallel, 181
dispositivo Printer, 363
dispositivo Serial, 214
dispositivo Timer, 426
dispositivo TrackDisk, 449
CloseLibrary, funzione, 151
CMD_CLEAR, comando, 19
dispositivo Audio, 108
dispositivo Console, 288
dispositivo Gameport, 337
dispositivo Keyboard, 311
dispositivo Serial, 218
dispositivo TrackDisk, 451
CMD_FLUSH, comando, 19
dispositivo Audio, 109
dispositivo Input, 250
dispositivo Narrator, 158
dispositivo Parallel, 186
dispositivo Printer, 369
dispositivo Serial, 219
CMD_INVALID, comando, 19
CMD_READ, comando, 19
dispositivo Audio, 111
dispositivo Clipboard, 400
dispositivo Console, 289
dispositivo Narrator, 159
dispositivo Parallel, 187
dispositivo Serial, 220
dispositivo TrackDisk, 458
CMD_RESET, comando, 19
dispositivo Audio, 113
dispositivo Clipboard, 402
dispositivo Input, 251
dispositivo Keyboard, 312
dispositivo Narrator, 161
dispositivo Parallel, 189
dispositivo Printer, 370
dispositivo Serial, 223
CMD_START, comando, 19
dispositivo Audio, 114
dispositivo Input, 252
dispositivo Narrator, 162
dispositivo Parallel, 190
dispositivo Printer, 370
dispositivo Serial, 224
CMD_STOP, comando, 19
dispositivo Audio, 116
dispositivo Input, 253
dispositivo Narrator, 163
dispositivo Parallel, 191
dispositivo Printer, 371
dispositivo Serial, 225
CMD_UPDATE, comando, 19
dispositivo Audio, 118
dispositivo Clipboard, 403
dispositivo TrackDisk, 461
CMD_WRITE, comando, 19
dispositivo Audio, 119
dispositivo Clipboard, 404
dispositivo Console, 293
dispositivo Narrator, 163
dispositivo Parallel, 192
dispositivo Printer, 372
dispositivo Serial, 226
dispositivo TrackDisk, 462
CmpTime, funzione, 427
Coda, XXVIII, 5
alla reply port, 2-4, 11
alla request port, 2-4, 11
Codici di tastiera grezzi, conversione, 286, 303
Comandi estesi, ETD_, 441
Comandi, invio dei

dispositivo Audio, 90
dispositivo Clipboard, 392
dispositivo Console, 272
dispositivo Gameport, 331
dispositivo Input, 242
dispositivo Keyboard, 306
dispositivo Narrator, 140
dispositivo Parallel, 174
dispositivo Printer, 355
dispositivo Serial, 202
dispositivo Timer, 421
dispositivo TrackDisk, 441
Command Line Interface, CLI, 351
Complex Interface Adapter, CIA, 417
Condizioni di EOF, 180, 212
Condizioni di trigger, 339, 345
Connettori. *Vedere* Pin, disposizione dei
Console, comandi del dispositivo per, 272, 288-300
azzerare il buffer di lettura, 288
impostare la mappa di tastiera, 299
impostare la mappa di tastiera di default, 298
leggere caratteri dalla tastiera, 289
leggere la mappa di tastiera, 297
leggere la mappa di tastiera di default, 296
scrivere caratteri sulla finestra, 293
Console, dispositivo, 267-300
comandi specifici del, 296
comandi standard del, 288
conversione dei codici di tastiera grezzi, 286
CSI, Control Sequence Introducer, 291
finestre di Intuition e, 267
Intuition e, 267
lettura e scrittura, operazioni di, 269
mappa di tastiera, 297, 299
mappa di tastiera di default, 296, 298
strutture di I/O del, 273
Console, funzioni del dispositivo per, 281
aprire l'unità, 284
chiudere l'unità, 283
convertire i codici grezzi, 286
gestire gli eventi di input, 281
Console, strutture del dispositivo, 273
Control Sequence Introducer, CSI, 291
ConUnit, struttura, 275, 284
CreateExtIO, funzione, 66, 473
CreatePort, funzione, 67, 474
CreateStdIO, funzione, 69, 475
CreateTask, funzione, 39, 70, 476
CSI, Control Sequence Introducer, 291
Custom, struttura, 94, 98
Cut & paste, 385-386

D

Dati, scambio di, 4
Dato campione, 79-82
DeleteExtIO, funzione, 71, 477
DeletePort, funzione, 71, 477
DeleteStdIO, funzione, 73, 478
DeleteTask, funzione, 74, 478
Device, struttura, 2
Dischi
accessibili per l'Amiga, 437
formato dei, 439
TrackDisk, interazioni con i, 439-441
Dispositivi hardware di controllo degli input, 325-329
Dispositivi, funzioni dei, 20
Dispositivo, 17
accesso esclusivo e condiviso al, 13
classi delle richieste di I/O, 6
code e loro comportamento, 11,
comandi immediati, 49
comandi standard del, 19
concetti di programmazione, XXVII
Device, struttura, 2
funzioni di accesso al, 57
funzioni standard del, 20
gestione del, 37
I/O accodato, 6
I/O veloce, 8
interazione dei task con il, 9
IORequest, struttura per il, 4, 24
IOStdReq, struttura per il, 4, 26
libreria del, 1
Message, struttura per il, 30
MsgPort, struttura per il, 28
request port del, XXVIII, 2, 11
residenti su ROM e su disco, XXVII
richieste di I/O, 1-31
richieste di I/O e code alle reply port, 2-4
unità del, 51
Unit, struttura per il, 2, 27
DMA, Direct Memory Access, 79, 99
DoIO, funzione, 4, 7, 44, 60
DVORAK, standard di tastiera, 271

E

Effetto vibrato, 100
EOF, caratteri di, 171, 176-177, 179, 199, 204-206, 209
EOF, caratteri speciali di, 181, 213
EOF, condizioni di, 180, 212
Errore, codici di, XXX
del dispositivo Audio, 101
del dispositivo Narrator, 148
del dispositivo Parallel, 180
del dispositivo Serial, 211
del dispositivo TrackDisk, 447
operazioni d'accesso, 14
standard, 102
ETD_, comandi estesi, 441
ETD_CLEAR, comando, 451
ETD_FORMAT, comando, 452
ETD_MOTOR, comando, 454
ETD_RAWREAD, comando, 455
ETD_RAWWRITE, comando, 457
ETD_READ, comando, 458
ETD_SEEK, comando, 460

ETD_UPDATE, comando, 461
ETD_WRITE, comando, 462
Eventi di input
  catena degli, 237-239
  creazione delle funzioni di gestione degli, 239-241
  dispositivo Console ed, 267
  flusso degli, 237-238
  funzioni di gestione degli, 237
  gestione degli, 239, 281
  grezzi e preelaborati, 271
  priorità delle funzioni di gestione degli, 239-241
Exec, funzioni di supporto alla libreria
  CreateExtIO, funzione, 66, 473
  CreatePort, funzione, 67, 474
  CreateStdIO, funzione, 69, 475
  CreateTask, funzione, 70, 476
  DeleteExtIO, funzione, 71, 477
  DeletePort, funzione, 71, 477
  DeleteStdIO, funzione, 73, 478
  DeleteTask, funzione, 74, 478
  NewList, macro, 75
Expunge, funzione di libreria, 64

F

FIFO, First In, First Out, 42
File Clipboard, 385
File comandi per stampante, 351
Filtro anti-aliasing, 96
Flag, XXX, 13
  del dispositivo Parallel, 178
  del dispositivo Serial, 208
  della struttura Unit, 27
  MsgPort, della struttura, 28
  programmazione dei, 40
  struttura di I/O, 26, 53, 97
Floppy disk. *Vedere* Dischi
Flusso degli eventi di input, 237-238
Forma d'onda, array della, 79-82
FreeSignal, funzione, 72
Frequenza del suono, 81
Frequenza di clock, 333
Funzioni di gestione degli eventi di input, 237-241
  priorità delle, 239-241
Funzioni di gestione del reset da tastiera, 303, 313, 319-320
Funzioni di supporto alla libreria Exec
  CreateExtIO, funzione, 66, 473
  CreatePort, funzione, 67, 474
  CreateStdIO, funzione, 69, 475
  CreateTask, funzione, 39, 70, 476
  DeleteExtIO, funzione, 71, 477
  DeletePort, funzione, 71, 477
  DeleteStdIO, funzione, 73, 478
  DeleteTask, funzione, 74, 478
  NewList, macro, 75

G

Gameport, comandi del dispositivo per, 331
  accertare il tipo di dispositivo hardware, 338
  accertare le condizioni di trigger, 339
  cambiare il tipo di dispositivo hardware, 344
  cambiare le condizioni di trigger, 345
  leggere gli eventi di input, 340
Gameport, dispositivo, 267, 325-345
  buffer interno del, 327, 337, 341
  comandi specifici del, 338
  comandi standard del, 337
  come generatore di eventi di input, 238
  dispositivi hardware previsti dal, 328-329
  dispositivo Keyboard e, 303
  elaborazione degli eventi di input condotta dal, 327
  funzionamento del, 327
  unità del, 327
  vedere anche Console, dispositivo
Gameport, funzioni del dispositivo, 334
Gameport, strutture del dispositivo, 332
GamePortTrigger, struttura, 332
GetMsg, funzione, 4, 7, 45
GPD_ASKCTYPE, comando, 338
GPD_ASKTRIGGER, comando, 339
GPD_READEVENT, comando, 340
GPD_SETCTYPE, comando, 344
GPD_SETTRIGGER, comando, 345
Graphics, libreria, 275

H

Handshaking, protocollo di, 224, 226

I

IDCMP, Intuition's Direct Communications Message Ports, 237
Identificatore del clip, 388
IFF, standard, 386, 398, 400, 406
INCLUDE, file, XXIII, XXIV, XXX, XXXIV, 31, 53
IND_ADDHANDLER, comando, 254
IND_REMHANDLER, comando, 255
IND_SETMPORT, comando, 256
IND_SETMTRIG, comando, 257
IND_SETMTYPE, comando, 259
IND_SETPERIOD, comando, 260
IND_SETTRESH, comando, 262
IND_WRITEEVENT, comando, 263
Input, comandi del dispositivo per, 242
  aggiungere eventi di input alla catena, 263
  aggiungere funzioni di gestione degli input, 254
  attivare l'unità 0, 252
  bloccare l'unità 0, 253
  determinare il tipo di dispositivo hardware di controllo, 259
  determinare la periodicità degli eventi temporizzati, 260
  determinare la porta giochi del mouse, 256
  determinare la soglia degli eventi temporizzati,

262
determinare le condizioni di trigger per il mouse, 257
effettuare il reset dell'unità 0, 251
eliminare eventi di input, 250
rimuovere funzioni di gestione degli input, 255
Input, dispositivo, 237-263
accesso al, 249
comandi specifici del, 254
comandi standard del, 250
dispositivo Console e, 267
dispositivo Gameport e, 327, 343
dispositivo Keyboard e, 303
dispositivo Timer e, 415
dispositivo TrackDisk e, 438
funzionamento del, 238
funzioni di gestione degli eventi di input, 239
gestione degli eventi condotta dal, 238-239
strutture di I/O previste dal, 243
Input, funzioni del dispositivo, 247-249
Input, strutture del dispositivo, 243
Input, task di, 237
InputEvent, struttura, 244
dispositivo Keyboard e, 303
INQ/ACK, protocollo, 206
Interfaccia parallela Centronics, specifiche dei pin, 172
Interfaccia seriale RS-232C, specifiche dei pin, 201
Interrupt di vertical-blanking, 415
Interrupt, struttura, 243
Intuition, finestre di, 20, 267
dispositivo Console e, 267, 284
dispositivo Input e, 237
parametri per le, 275
Intuition, libreria, 239-241, 267
Intuition, priorità nell'elaborazione degli eventi di input, 241
I/O, richieste di, 2-4, 31-32
asincrono, 41
classi delle, 6-8, 53
elaborazione in modo immediato delle, 49
gestione multitasking delle, 15, 415
modi di accesso, 13
multiple, 47, 54
più reply port e più unità, 9
preparazione ed elaborazione delle, 47, 51
sincrono, 45
I/O, strutture di, XXVIII
Io_Data, parametro, 4, 24
IOAudio, struttura, 94
IOClipReq, struttura, 394
IODRPReq, struttura, 362
IOExtPar, struttura, 177
IOExtSer, struttura, 206
IOExtTD, struttura, 443
IOF_QUICK, flag, 8
IOPar, struttura, 176
IOPArray, struttura, 176
IOPrtCmdReq, struttura, 360
IORequest, struttura, 4, 24
IOStdReq, struttura, 4, 26
IOTArray, struttura, 205

J

Joystick
assoluto, 329
relativo, 329

K

KBD_ADDRESETHANDLER, comando, 313
KBD_READEVENT, comando, 315
KBD_READMATRIX, comando, 317
KBD_REMRESETHANDLER, comando, 319
KBD_RESETHANDLERDONE, comando, 320
Kermit, protocollo, 208
Keyboard, comandi del dispositivo per, 305, 311-320
aggiungere una funzione di gestione del reset, 313
finire l'esecuzione della funzione di reset, 320
leggere la matrice dello stato dei tasti, 317
leggere un evento da tastiera, 315
rimuovere una funzione di gestione del reset, 319
Keyboard, dispositivo, 303-320
buffer del, 303-304
comandi specifici del, 338
comandi standard del, 337
come generatore di eventi di input, 238
dispositivo Input e, 237
elaborazione degli eventi di input, 304
flusso di eventi al, 271
funzionamento del, 304
stato dei tasti, 318
Keyboard, funzioni del dispositivo, 334
Keyboard, strutture del dispositivo, 332
KeyMap, struttura, 278
KeyMapNode, struttura, 280
KeyMapResource, struttura, 280
Kickstart, disco del, XXXI

L

La, nota fondamentale, 81
Lattice, compilatore C, XXXI
Lib_OpenCnt, parametro, 14
Library, struttura, 2
Librerie, funzioni di gestione delle, 151, 155
List, struttura, 24
Liste, gestione delle, 3

M

Mappa di tastiera, 271
Memoria, XXIII, XXVI, XXVII
eventi di input e, 237
funzione RemDevice e, 64
task e, 17
vedere anche RAM
Message port, XXVIII, 2, 28

segnali alla, 8, 29-30
vedere anche Reply port del task
Message, struttura, XXVIII, 3-4, 30
Messaggio, definizione di, XXVIII, 2-4
mittente e destinatario del, 3-4, 23
MFM, Modified Frequency Modulation, formato, 455, 457
MIDI, Musical Instrument Digital Interface, interfaccia, 209
Modem, 199
Modo immediato, 49
dispositivo Gameport, 331
dispositivo Input, 242
dispositivo Parallel, 174
Modulazione di frequenza, 100
Mouse, 327
come generatore di eventi di input, 238
dispositivo Console e, 267
dispositivo Gameport e, 325
Mouth_rb, struttura, 146
MsgPort, struttura, XXVIII, 2, 23-24, 28
segnali, 8, 29-30
Multitasking, XXVI, 15, 79, 415

N

Narrator, comandi del dispositivo per, 158
attivare l'unità, 162
avviare la riproduzione vocale, 163
bloccare l'unità, 163
effettuare il reset dell'unità, 161
eliminare richieste di I/O, 158
ottenere sagome della bocca, 159
Narrator, dispositivo, 139-163
codici d'errore del, 148
comandi standard del, 158
elaborazione del testo, 139
priorità dei canali audio, 144
strutture previste dal, 142
Narrator, funzioni del dispositivo per, 149-156
aprire l'unità, 152
aprire la libreria Translator, 155
chiudere la libreria Translator, 151
chiudere l'unità, 149
tradurre una stringa in fonemi, 156
Narrator, strutture del dispositivo, 142
Narrator_rb, struttura, 143
NewList, macro, 75
Nibble, 82
Node, struttura, 3, 24-25
NTSC e PAL, differenze fra macchine, 81

O

OpenDevice, funzione, XXVII, 20
dispositivo Audio, 104
dispositivo Clipboard, 398
dispositivo Console, 284
dispositivo Gameport, 335
dispositivo Input, 249
dispositivo Keyboard, 309
dispositivo Narrator, 152
dispositivo Parallel, 183
dispositivo Printer, 364
dispositivo Serial, 215
dispositivo Timer, 428
dispositivo TrackDisk, 450
OpenLibrary, funzione, 155

P

PAL e NTSC, differenza fra macchine, 81
Parallel, comandi del dispositivo per, 186-195
attivare il dispositivo, 190
bloccare il dispositivo, 191
effettuare il reset del dispositivo, 189
eliminare le richieste di I/O, 186
impostare i parametri di funzionamento, 195
inviare dati attraverso la porta parallela, 192
leggere dati provenienti dalla porta parallela, 187
leggere il registro di stato della porta, 193
Parallel, dispositivo, 171-195
codici d'errore del, 180
comandi specifici del, 193
comandi standard del, 186
condizioni di EOF, 180
dispositivo Printer e, 349
dispositivo Serial e, 349
EOF, condizioni di, 180
operazioni di lettura e scrittura, 171
strutture previste dal, 176
Parallel, funzioni del dispositivo, 181-185
Parallel, strutture del dispositivo, 176
Parallela Centronics, specifiche dei pin dell'interfaccia, 172
Parametri
chiave d'assegnazione, 87, 95
ConUnit, struttura, 275
Message, struttura, 30
RastPort, struttura, 277
reply port, XXVIII, 2
SRE e RRE, sequenze, 291
Unit, struttura, 27
PDCMD_QUERY, comando, 193
PDCMD_SETPARAMS, comando, 195
Periodo del suono, 81
Periodo di campionamento del suono, 79-82
Periodo, registro di, 79-82
Pin, disposizione dei
connettore disk drive, 439
connettore interfaccia parallela, 172
connettore interfaccia seriale, 201
connettore porta giochi, 326
PRD_DUMPRPORT, comando, 374
PRD_PRTCOMMAND, comando, 377
PRD_QUERY, comando, 379
PRD_RAWWRITE, comando, 380
Printer, comandi del dispositivo per, 355, 369
attivare l'unità, 370
bloccare l'unità, 371

effettuare il reset dell'unità, 370
eliminare le richieste di I/O, 369
inviare una particolare sequenza di comando, 377
inviare un testo alla stampante, 372
inviare un testo grezzo alla stampante, 380
ottenere dati sullo stato della stampante, 379
stampare una bitmap, 374
Printer, dispositivo, 349-380
funzionamento del, 351
PrinterIO, unione del, 358
sequenze di controllo, 353
Printer, funzioni del dispositivo, 363
Printer, strutture del dispositivo, 359
PrinterIO, unione, 358
Priorità d'assegnazione dei canali audio, 85
Protezione dei canali audio, 85, 129
Protocolli di trasmissione
INQ/ACK, 206
Kermit, 208
Xmodem, 208
XON/XOFF, 206
Protocollo di handshaking, 224, 226

Q

QuickIO, richieste con il, 8
dispositivo Audio, 91
dispositivo Clipboard, 392
dispositivo Console, 272
dispositivo Gameport, 331
dispositivo Input, 242
dispositivo Keyboard, 306
dispositivo Narrator, 140
dispositivo Parallel, 174
dispositivo Printer, 355
dispositivo Serial, 202
dispositivo Timer, 421
dispositivo TrackDisk, 441
QWERTY, standard di tastiera, 271

R

RAM, XXIII, XXVI, XXVII, 64, 72-73, 79, 95, 417
struttura InputEvent e, 237, 241-242
RastPort, struttura, 277
RawKeyConvert, funzione, 286
Registro di campionamento, 79-82
Registro di stato della
porta parallela, 173
porta seriale, 200
RemDevice, funzione, 2, 64
Remove, funzione, 4, 7, 44
Reply port, coda alla, 12
Reply port del task, XXVIII, 2, 12
ReplyMsg, funzione, 2-5
Request port, coda alla, 11
Request port del dispositivo, XXVIII, 2, 11
Reset Raw Events (RRE), sequenza, 291
Richieste di I/O asincrono, 41
BeginIO e, 4, 7, 44, 58
SendIO e, 4, 7, 44, 61
Richieste di I/O sincrono, 45
BeginIO e, 4, 7, 44, 58
DoIO e, 4, 7, 44, 60
RS-232C, interfaccia seriale, 201

S

SatisfyMsg, struttura, 396
SDCMD_BREAK, comando, 228
SDCMD_QUERY, comando, 230
SDCMD_SETPARAMS, comando, 231
Segnali, 8, 29-30, 44
SendIO, funzione, 4, 7, 44, 61
Sequenze escape, 351
Serial, buffer di lettura interno del dispositivo, 199
azzeramento del, 218
dimensione di default, 199
Serial, comandi del dispositivo per, 218-231
acquisire i dati attraverso la porta seriale, 220
attivare l'unità, 224
azzerare il buffer interno di lettura, 218
bloccare l'unità, 225
effettuare il reset dell'unità, 223
eliminare le richieste di I/O, 219
impostare i parametri di controllo, 231
rilevare lo stato della porta seriale, 230
trasmettere i dati attraverso la porta seriale, 226
trasmettere il segnale di break, 228
Serial, dispositivo, 199-231
codici d'errore del, 211
comandi specifici del, 228
comandi standard del, 218
condizioni di EOF del, 212
flag previsti dal, 208
operazioni di lettura e scrittura del, 199
strutture del, 204
Serial, funzioni del dispositivo, 214
Serial, strutture del dispositivo, 204
Seriale RS-232C, specifiche dei pin dell'interfaccia, 201
Set Raw Events (SRE), sequenza, 291
Sincrono, richieste di I/O, 45
Sottrazione dei canali, 85-86
SRE e RRE, sequenze, 291
Stampanti previste dall'Amiga, 349
Stringa di fonemi, 139, 156
Stringa inglese, 139
SubTime, funzione, 430
Suono, 79
attenuazione del, 80
differenze fra macchine NTSC e PAL, 81
frequenza del, 81
nota fondamentale La, 81
periodo del, 81
periodo di campionamento del, 81
tick di sistema, 81

T

Task
  di gestione dei dispositivi, 37
  dispositivo Console e, 267
  interazioni fra dispositivi e, 1
Task, coda alla reply port del, 12
Task di input, 237
Task, priorità del, 15
Task, reply port del, XXVIII, 2, 12
Tastiera DVORAK, 271
Tastiera QWERTY, 271
Tavola di eccezioni nella trasformazione in fonemi, 139
TD_ADDCHANGEINT, comando, 463
TD_CHANGENUM, comando, 464
TD_CHANGESTATE, comando, 465
TD_FORMAT, comando, 452
TD_GETDRIVETYPE, comando, 466
TD_GETNUMTRACKS, comando, 467
TD_MOTOR, comando, 454
TD_PROTSTATUS, comando, 467
TD_RAWREAD, comando, 455
TD_RAWWRITE, comando, 457
TD_REMCHANGEINT, comando, 468
TD_REMOVE, comando, 469
TD_SEEK, comando, 460
TDU_PublicUnit, struttura, 445
Tempo di sistema, 415
Testo, eventi di input e, 271
Tick di sistema, 81
Timer, aritmetica del dispositivo, 415, 418-421
Timer, comandi del dispositivo per, 421, 431
  impostare il tempo di sistema, 434
  inviare una richiesta di temporizzazione, 431
  ottenere il valore del tempo di sistema, 432
Timer, dispositivo, 415-434
  accuratezza, 415, 417
  comandi specifici del, 431
  come generatore di eventi di input, 238
  compensazione degli errori di temporizzazione, 415
  differenze tra versione europea e americana, 417
  dispositivo Input e, 237
  funzionamento del, 415
  operazioni aritmetiche con il tempo, 415
  periodo di tempo, 415
  unità del, 417
  UNIT_MICROHZ, costante, 417
  UNIT_VBLANK, costante, 417
Timer, funzioni del dispositivo per, 424
  aprire l'unità, 428
  chiudere l'unità, 426
  confrontare due tempi, 427
  sommare due tempi, 424
  sottrarre due tempi, 430
Timer, strutture del dispositivo, 422
Timerequest, struttura, 423
Timeval, struttura, 423
TR_ADDREQUEST, comando, 431
TR_GETSYSTIME, comando, 434
TR_SETSYSTIME, comando, 434
TrackDisk, comandi del dispositivo per, 441, 451-469
  accertare la presenza di un disco nell'unità, 465
  accertare lo stato della linguetta di protezione, 467
  aggiungere una funzione di gestione dell'interrupt prodotto dalla rimozione e dall'inserimento del disco, 463
  attivare l'interrupt di rimozione del disco, 469
  attivare/disattivare il motore, 454
  azzerare i buffer di un'unità, 451
  eliminare la funzione di gestione dell'interrupt prodotto dalla rimozione e dall'inserimento del disco, 468
  formattare il disco, 452
  leggere da disco dati decodificati, 458
  leggere da disco dati non decodificati, 455
  memorizzare su disco dati codificati, 462
  memorizzare su disco dati non codificati, 457
  muovere la testina da traccia a traccia, 460
  ottenere il numero di rimozioni e inserimenti di dischi, 464
  ottenere il numero di tracce previsto dal disk drive, 467
  ottenere il tipo del disk drive associato all'unità, 466
  salvare il contenuto dei buffer, 461
TrackDisk, dispositivo, 437-469
  buffer del, 437
  codici d'errore del, 447
  comandi specifici e standard del, 451
  come generatore di eventi di input, 238
  dispositivo Input e, 237
  dispositivo Keyboard e, 303
  ETD_, comandi estesi del, 441
  funzionamento del, 437
  interazioni con i dischi del, 439-441
  strutture del, 442
TrackDisk, funzioni del dispositivo, 449
TrackDisk, strutture del dispositivo, 442
Translate, funzione, 156
Translator, libreria, 139
  accesso alla, 151, 155
TranslatorBase, variabile globale, 155, 158
Trigger, condizioni di, 339, 345

U

Unit, struttura, 2, 27
  dispositivo Console e, 267
Unit_OpenCnt, parametro, 14

V

Variabile globale, TranslatorBase, 155, 158
Vertical-blanking, interrupt di, 415
Voce
  parametri della, 142
  riproduzione vocale, 139

Voice Synthesis, libreria interna, 139
Voice, sotto-struttura, 146
Volume
  modifica del, 131
  valori possibili del, 80

W
Wait, funzione, 4, 8, 44
WaitIO, funzione, 4, 7, 44, 61
WaitPort, funzione, 4, 43-44
WCS, Write Control Store, XXVII
Window, struttura, 267

X
XON/XOFF, protocollo, 206
XModem, protocollo, 208

Questo volume è stato stampato nel mese di settembre 1989
presso gli stabilimenti della Alberto Matarelli S.p.A.
Stampato in Italia – Printed in Italy

PROGRAMMARE L'AMIGA

VOLUME II

Programmare l'Amiga Volume II costituisce un approfondito manuale di riferimento per utilizzare i dispositivi di I/O dell'Amiga 500, 1000 e 2000. I dispositivi sono gli strumenti software che permettono di generare suoni stereofonici, di far parlare l'Amiga, di utilizzare la stampante, il modem, la porta seriale e quella parallela, i file clipboard, le porte giochi...

Completamente aggiornato alla versione europea dell'Amiga, questo volume illustra dettagliatamente i comandi, le funzioni, le architetture e i principi di funzionamento dei dodici dispositivi di I/O dell'Amiga. L'edizione italiana è stata redatta apportando aggiornamenti e correzioni.

Due capitoli introduttivi trattano la gestione dei dispositivi di I/O in generale, con considerazioni pratiche, metodi di accesso e tecniche di programmazione che valgono per tutti i dispositivi. Vengono analizzati i concetti di accodamento al dispositivo, di accodamento al task, di formulazione di una richiesta di I/O, e viene illustrato come utilizzare le funzioni di supporto alla libreria Exec. Seguono dodici capitoli dedicati ai dispositivi:

- Audio
- Serial
- Keyboard
- Clipboard
- Narrator/libreria Translator
- Input
- Gameport
- Timer
- Parallel
- Console
- Printer
- TrackDisk

Il testo è accompagnato da numerose illustrazioni e diagrammi sul funzionamento dei dispositivi e delle loro funzioni, sull'elaborazione dei comandi, e sulle strutture di dati necessarie. Il risultato è una fonte di riferimento sull'Amiga che non ha eguali.

Questo libro è indispensabile per chiunque desideri (per lavoro o per hobby) conoscere i dispositivi dell'Amiga e utilizzarli nello sviluppo di programmi sofisticati. Si rivolge espressamente a programmatori in linguaggio C e Assembly, ma è anche utile a tutti coloro che da Amiga BASIC desiderano arrivare a una migliore comprensione del sofisticato sistema dei dispositivi di I/O di cui l'Amiga è dotato.

L'AUTORE

Eugene P. Mortimore è presidente della Micro Systems Analysis, Inc., un'azienda di consulenza informatica specializzata nei computer Amiga e IBM. È autore di numerosi libri sui microcomputer e vive a Bethel Park, in Pennsylvania (USA).

ISBN 88-7803-005-8

Lire 70.000

Eugene P.
Mortimore

Programmare l'Amiga Volume II

iHT
GRUPPO
EDITORIALE

MUSEO DEL COMPUTER
Fondazione ONLUS
Sede legale
Via Costantino Perazzi 22
28100 NOVARA
Tel 0321 1856032
www.museodelcomputer.org
info@museodelcomputer.org

C.F. 94064520037
Registro P.G. 237

Documento redatto da Alberto Rubinelli in data 06-01-2012 Revisione 1.4