Default

COLLABORATORS

	TITLE : Default	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Default

1.1 L’hardware dell’Amiga

L’hardware dell’Amiga

Il cuore del sistema come ben sapete è il 680x0 della Motorola; scelta decisamente giusta in quanto questo microprocessore possiede una serie di caratteristiche (trap, modo supervisore e utente ecc.) che lo rendono particolarmente efficace per il multitasking; ed è ben conosciuta la caratteristica unica dell’Amiga: i coprocessori; infatti il microprocessore è attorniato da una serie di coprocessori, ognuno dedicato ad un operazione ben specifica, che sollevano quest’ultimo dal compito di occuparsi di determinati problemi (spesso di ingresso/uscita) per dedicarsi completamente a quelli di calcolo. Come avrete sentito più volte questi processori hanno dei nomi piuttosto pittoreschi: Denise (sostituito nelle macchine AGA da Lisa), Fat Agnus (attualmente Alice), Paula, CIA (e adesso sul CD32 se ne aggiunto uno nuovo: Akiko); trascurando il perché di questi nomi tutti femminili (io veramente mi preoccuperei se venisse prodotto un integrato con un nome del tipo: Ugo!) andiamo ad analizzarne le caratteristiche. Lisa è il processore grafico, cioè colui che crea tutte le risoluzioni, determina tutte le frequenze video, insomma visualizza tutto sullo schermo; Alice forse è il più importante di tutti ed è composto da più processori messi insieme utilizzando una definizione in VLSI (Very Large Scale Integration = Integrazione a Scala Molto Grande) per compattezza e per minimizzare i costi; contiene innanzitutto il Blitter il cui nome proviene da BLIT (BLock Image Transfer = Trasferimento di Immagine a Blocchi); scopo di questo integrato è spostare blocchi di memoria rettangolari (che nella stragrande maggioranza si tratta di blocchi grafici) con operazioni logiche e ad una velocità impressionante; permette anche di tracciare linee ed eseguire il filling (riempimento) grafico; potete ben immaginare che essendo tutto ciò realizzato via hardware risulta velocissimo; in realtà grazie alla funzione logica programmabile che opera sui dati in tempo reale, se ne fà degli utilizzi più svariati e non solo per la grafica (decodifica MFM, cancellazione di memoria ecc.). Altro integrato contenuto in Alice è il Copper; questo è un velocissimo processore RISC (Reduce Instructions Set Chip = Chip con set di istruzioni ridotto) che permette solo 3 istruzioni e risulta sincronizzato al pennello video! Vale a dire che il programma del Copper può attendere una precisa posizione di quest’ultimo (e non solo in verticale) per eseguire determinate operazioni; dovete a lui la possibilità di slittare gli schermi uno sopra l’altro, infatti quando si raggiunge una determinata posizione

verticale dello schermo il copper cambia i registri di Lisa inserendo i parametri del secondo schermo, e tornando all'inizio del video ripristina quelli vecchi. Fa parte di Alice anche la circuteria per la gestione e temporizzazione dei canali DMA che vedremo dopo; arriva quindi Paula il processore che si occupa sostanzialmente della generazione sonora dalle caratteristiche note a tutti. Infine ci sono due CIA (Complex Interface Adapter = Adattatore complesso di interfaccia) siglati numericamente come 8520 e che erano presenti anche nel C64! Si tratta degli integrati che regolano il flusso dei dati con l'esterno; infatti tramite loro si gestiscono drive, porta seriale e parallela, tastiera e le porte joystick. Ma non è finito qui, poiché occorre indicare dove i dati a cui questi integrati fanno riferimento vengono memorizzati (cioè dove si trovano i bitplanes contenenti i dati dello schermo per Lisa, o i suoni per Paula); e qui è presente un'altra caratteristica unica dell'Amiga, infatti tutti i dati sono conservati nella stessa memoria, accedibile anche dal microprocessore centrale; questo è sicuramente positivo, infatti su altre piattaforme se viene resa disponibile un determinato quantitativo di memoria su una scheda, ad esempio grafica, e se ne viene utilizzata solo una parte, il resto non può essere disponibile ad altre risorse del sistema; ma come è possibile che tutti gli integrati accedano contemporaneamente alla stessa memoria? Tutto ciò è in parte merito dei canali DMA (Direct Memory Access = Accesso Diretto della Memoria) che permettono a ciascun processore di prelevare i dati dalla RAM senza l'ausilio della CPU; ma ciò non esaurisce il problema, perché come tutti i "vecchi" (che non erano vecchi prima dell'avvento di Amiga) libri di informatica descrivono, quando il canale DMA accede alla memoria centrale, lascia il microprocessore libero dall'oneroso compito di trasferire i dati e dedicarsi al programma, senza però concederne l'accesso alla memoria; bene il grandioso cast di ingegneri di Amiga ha ben pensato di partizionare il tempo dedicato ad ogni canale DMA (sono 28 in tutto) e il microprocessore; mentre fra i canali DMA questo non comporta quasi nessuna limitazione (tranne per quelli del Blitter e del Copper) perché si conoscono a priori le esigenze di tempo di ciascuno, lo stesso non si può dire per il microprocessore che in un buon numero di casi è costretto ad aspettare il prossimo slot time (così si chiama il periodo di tempo concesso ad ogni canale) se quello attuale è occupato da un DMA; l'idea come era congeniata per i primi modelli funzionava bene: infatti il 68000 impiegava uno slot di tempo per eseguire il "fetch" dell'istruzione (cioè prelevamento dell'istruzione e degli operandi dalla memoria) e per l'accesso alla memoria e quello successivo per la fase di "execute" dell'istruzione in cui non vi erano quasi mai accessi alla memoria; per cui si è cercato di inserire tutti i possibili accessi dei DMA negli slot pari (in cui il microprocessore era nella fase di execute) e lasciare liberi quelli dispari per il microprocessore; però anche nel sistema iniziale il microprocessore attendeva poiché in caso di utilizzo di risoluzioni elevate gli slot concessi ai DMA del Denise sconfinavano in quelli dispari; la maggior parte degli altri canali DMA (sprite, disco, audio) erano posizionati in maniera fissa e se non venivano utilizzati (perché in quel momento non si legge dal disco ad esempio) potevano essere usati dagli altri che risultano dinamici, cioè che possono utilizzare qualunque slot libero quando necessario; questi sono il microprocessore, il Blitter ed il Copper; ma con quale criterio viene scelto quello piuttosto di quell'altro per l'assegnazione? Bene, per questo sono previsti dei registri di priorità; ad esempio il Blitter ha normalmente priorità più alta del microprocessore per cui se il Blitter in un dato momento opera, al microprocessore vengono assegnati un numero minimo di slot time. La memoria che viene condivisa in questo modo, viene denominata CHIP e quindi risulta

più lenta da sfruttare (vista dal microprocessore); ma può essere addizionata al sistema altra memoria che viene utilizzata solo dal microprocessore centrale e che quindi può essere sfruttata a pieno da quest'ultimo; questa viene denominata FAST (il 68020 di un A1200 con FAST memory gira circa il doppio più veloce rispetto ad uno senza). Fanno eccezione a quanto detto fino adesso, i due CIA con cui il microprocessore deve interagire direttamente; per cui sia parallela, che seriale, tastiera e porte joystick non dispongono di canali DMA (mentre il disk drive si) e devono essere gestiti dal microprocessore mediante interrupt.

1.2 Corso di programmazione su Amiga

Corso di programmazione su Amiga

di Giuseppe Ligorio

Questo è un piccolo corso su come programmare l'Amiga con sistema operativo v39; questa è la versione AmigaGuide di una parte (le prime 10 puntate) del corso "Impariamo a programmare l'Amiga" su Enigma Amiga RUN.
Viene richiesta semplicemente di avere la conoscenza su come si programma in C standard, un compilatore C (Lattice SAS ad esempio) e cosa molto importante bisogna stare attenti alla versione degli include utilizzati dal compilatore; di volta in volta verranno esaminate le routines più importanti del s.o. e verrà sempre indicata la versione del kickstart necessaria per utilizzare tali funzioni.

Sommario:
-

1.3 Introduzione alle librerie

Introduzione alle librerie

Tutte le librerie (tranne exec che come è detto è già operante) vanno aperte tramite un'apposita funzione di exec: OpenLibrary; ogni libreria è formata oltre che da una serie di informazioni, anche da una jump table, cioè da una tabella di salti che vi permetteranno di raggiungere la funzione desiderata; questo non ha comunque molta importanza, poiché da C le funzioni di libreria vengono chiamate come normali procedure, passando i parametri ove necessario e prelevando se ne vengono ritornati; ma lasciando l'approfondimento di tutto ciò quando verrà trattato exec, passiamo alla prima libreria che verrà secritta: la dos.library.

1.4 Il sistema operativo

Il sistema operativo

Come alcuni di voi avranno imparato, il compito del sistema operativo di un elaboratore è frapporsi fra l'utente, i programmi e l'hardware; in altri termini costituisce un'interfaccia in modo che dal punto di vista dei programmi facci apparire la macchina come una serie di device (dispositivi) che permettano di usufruire di quest'ultima senza utilizzare direttamente l'hardware; dall'altra parte deve convertire i comandi impartiti dai programmi ai device logici (definiti logici perché sono "immaginari" cioè non fisici) in una serie di istruzioni che pilotino correttamente l'hardware; tutto ciò non solo per massima flessibilità di utilizzo ma soprattutto, in caso di eventuale cambio di hardware (gli amighisti ne sanno qualcosa), basta semplicemente cambiare il s.o. senza mutare i programmi che così potranno girare correttamente anche su modelli diversi.
Il s.o. dell'Amiga è composto da diversi livelli di software per una migliore gestione della poderosa struttura hardware appena vista, ma anche per le caratteristiche potenti di cui è dotato (multitasking, interfaccia grafica in ambiente WIMP = Windows Icons Menù Puntator = finestre, icone, menù e puntatori ecc.).
I diversi livelli di Amiga, partendo dal più basso sono:
- Hardware
- Resource
- Device

- Library
- Workbench, CLI, Programmi
- Utente.

Il software di livello immediatamente superiore all'hardware è normalmente il device, che si occupa tramite comandi vari di pilotare l'hardware; alcuni devices sono: trackdisk.device, parallel.device, serial.device, audio.device ecc.. Non sempre il device è il software direttamente connesso, in alcuni casi a causa della difficoltà di gestione del dispositivo hardware, occorre un ulteriore livello che viene denominato resource; esempio il disk.resource, ricordo infatti che il drive viene pilotato direttamente senza controller hardware, come comunemente avviene in altri sistemi (e proprio per questo che l'Amiga può leggere altri formati di dischi). In altri casi invece non occorre nemmeno il device, per cui la library risulta connessa direttamente all'hardware, un esempio è la graphics.library.
La library o tradotto in italiano libreria, è una serie (una libreria appunto) di funzioni messe a disposizione dei programmi (e quindi a colui che li crea) per gestire tutta la macchina; esempi di libreria sono: exec.library, graphics.library, intuition.library, dos.library, asl.library ecc. A quanto detto exec.library costituisce un'eccezione, in quanto si trova praticamente a tutti i livelli (cioè può pilotare direttamente l'hardware, alcune librerie possono sfruttare sue funzioni, può pilotare direttamente librerie ecc.); per farvi capire quanto questa libreria sia importante sappiate che l'indirizzo in cui è contenuto il suo puntatore è l'unica cosa fissa dell'Amiga! Tutto il resto: device, librerie, variabili, programmi vengono allocati (fra l'altro proprio grazie a funzioni dell'exec) dove c'è disponibilità di memoria; insomma l'exec è colui che si occupa di far funzionare correttamente tutto l'Amiga.
Nel nostro viaggio partiremo dalle librerie (che costituiscono il 90% di quello che un programmatore Amiga deve conoscere) fino ai devices e resorces, e anche qualcosa sull'hardware che il s.o. non ci può dare.

1.5 La dos.library

La dos.library

Programmando in C la dos.library costituisce l'unica eccezione a quanto detto prima sulle librerie; infatti non vi è bisogno di aprirla perché l'apertura e la chiusura vengono automaticamente effettuati dal compilatore; questo per mantenere la compabilità con lo standard ANSI C (e quindi garantire la portabilità dei programmi); infatti lo standard prevede di gestire files senza bisogno di aprire e chiudere alcunché; apertura e chiusura che devono quindi essere realizzate in maniera trasparente al programmatore. Come avrete capito la dos.library mette a disposizione le funzioni per gestire il DOS (Disk Operating System = Sistema Operativo del Disco), cioè gestione dei files, directory, drive et similia. Iniziamo subito con la gestione dei files, ed in particolare con la funzione che permette di aprirne uno:

```
file = Open(nomefile,modalità);
```

dove nomefile è il puntatore alla stringa contenente il nome del file o del dispositivo (tipo CON: o RAW:) completo di path; modalità indica la modalità appunto con cui aprire il file e può valere MODE_OLDFILE, per aprire un file già esistente per la lettura e scrittura (se il file non

esiste verrà segnalato un
errore
), MODE_NEWFILE che crea un nuovo file
per lettura e scrittura (se il file esiste già verra inizializzato comunque); ed infine MODE_READWRITE che apre un file, sempre per operazioni sia di lettura che di scrittura in modalità spartita (ciò significa che lo stesso file potrà essere aperto anche da altri processi, mentre per le precedenti modalità il file veniva bloccato, cioè altri processi non possono accedervi fino a quando non viene chiuso dal programma che lo usa attualmente); file è il puntatore ad una struttura FileHandle che contiene tutte le informazioni relative al file aperto; informazioni che necessitano a tutte le altre funzioni che operano sui file, per cui tale puntatore dovrà essere passato a tutte le funzioni che verranno chiamate successivamente; se il valore ritornato da Open è nullo, allora il file non è stato aperto per qualche motivo, per cui si è verificato un errore; vediamo un esempio:

```
struct FileHandle *Fh;
  .
  .
if ((Fh = Open("ram:pippo",MODE_OLDFILE)) == NULL)
{
  printf("Non posso aprire il file pippo.\n");
  exit(1);
}
```

Dopo aver finito di utilizzare il file, occorre chiuderlo con la seguente funzione:

```
Close(file);
```

dove file è il puntatore a FileHandle ritornato dalla rispettiva funzione Open. La volta scorsa abbiamo parlato delle funzioni della dos.library per aprire e chiudere un file, le prossime istruzioni da esaminare riguardano la gestione di quest'ultimo; i files nell'AmigaDOS vengono gestiti sia ad accesso sequenziale che casuale (random); ciò significa che all'apertura del file, il cursore che segna la lettura è posizionato all'inizio di quest'ultimo e, man mano che si effettuano operazioni di lettura o di scrittura, quest'ultimo si posiziona alla fine della lunghezza dell'ultima operazione realizzata; in questa maniera si sfrutta un accesso sequenziale al file, esiste tuttavia un metodo per posizionarsi in un qualsiasi punto del file e utilizzare così un accesso casuale. La prima funzione da analizzare è Read che, come si può intuire dal nome, realizza l'operazione di lettura:

```
LungAttuale = Read(file,buffer,lunghezza);
```

dove file è il puntatore alla struttura FileHandle ritornata dalla funzione Open, buffer è un puntatore ad un buffer in memoria che conterrà i dati da leggere, e lunghezza il numero di bytes da leggere dal file; il parametro ritornato LungAttuale indica il numero di bytes effettivamente letti dall'operazione di Read; normalmente tale valore è positivo e serve a verificare che il numero di bytes letti equivale effettivamente a quello richiesto (in caso contrario si sarà raggiunto l'end-of-file); se tale numero vale zero significa che ci si trova già sull'end-of-file e quindi non è possibile leggere alcun byte; se tale valore risulta infine -1 significa che si è verificato un

```
                errore
                ; osserviamo immediatamente un
esempio:

struct FileHandle *file;
LONG len;
UBYTE buffer[1000];
  .
  .
len = Read(file,buffer,1000);
if (len < 0) printf("Errore\n");
else if (len < 1000) printf("Letti meno bytes, raggiunto EOF");
```

questo esempio tenta di leggere 1000 bytes dal file identificato dalla struttura FileHandle denominato 'file' e di inserirlo nel buffer denominato 'buffer' (che nella fattispecie è creato come vettore); se il valore ritornato len è negativo viene stampato su video un errore, se è inferiore a 1000 viene avvisato l'utente sempre con un messaggio.
La seconda istruzione da esaminare è Write che serve per la scrittura in un file; la sintassi è:

```
LungRitorno = Write(file,buffer,lunghezza);
```

dove 'file' ricopre il solito significato, 'buffer' è il puntatore al buffer contenente i dati da scrivere e 'lunghezza' indica il numero di byte da scrivere; 'LungRitorno' indica il numero di bytes realmente scritti sul file; anche in questo caso in presenza di errore il valore ritornato è -1; l'esempio viene riportato qui di seguito:

```
struct FileHandle *file;
LONG len;
UBYTE buffer[1000];
  .
  .
len = Write(file,buffer,1000);
if (len < 0) printf("Errore\n");
```

Le istruzioni Read e Write sono non bufferizzate, ciò significa che se leggo un solo byte alla volta il dos accederà ogni volta al file impiegando così molto tempo; una vecchia soluzione per risolvere questo problema era leggere un blocco di bytes riponendolo in un buffer e da lì prelevare i singoli caratteri; dalla versione 2.0 del sistema operativo (V36) sono previste però due funzioni che permettono di leggere e scrivere un byte alla volta con bufferizzazione implementata dal DOS: FGetC e FPutC. La prima preleva un carattere dalla poszione corrente del file e il secondo inserisce un carattere nella posizione corrente del file; la sintassi di queste è:

```
carattere = FGetC(file);
```

dove 'carattere' è il byte letto dal file puntato da 'file'.

```
carattere1 = FPutC(file,carattere2);
```

dove 'carattere2' è il byte da scrivere nel file puntato da 'file'; il valore ritornato 'carattere1', o è identico a carattere2 o corrisponde a EOF se si è verificato errore.

Dato che con queste operazioni viene effettuata una bufferizzazione, se dopo la chiamata di una di queste si ha intenzione di utilizzare un Read o un Write (che non sono bufferizzate), occorre effettuare un operazione di scarico del buffer: Flush;

```
successo = Flush(file);
```

dove il valore ritornato indica il successo dell'operazione.
Un'altra operazione importante legata a quelle bufferizzate è UnGetC che permette da inserire nel buffer un qualsiasi byte e che quindi verrà acquisito nella prossima operazione di lettura (se questa è bufferizzata);

```
valore = UnGetC(file,carattere);
```

'carattere' è il byte che viene inserito nel buffer di I/O; se questi vale -1 allora verrà reinserito nel buffer l'ultimo byte letto (sempre da operazione bufferizzata e vale a dire FGetC); il parametro ritornato 'valore', indica il carattere inserito nel buffer o FALSE se tale carattere non può essere inserito nel buffer.
Come abbiamo accennato, vi è la possibilità di accedere casualmente al file mediante un istruzione che ci permette di spostarci in un punto qualsiasi di quest'ultimo: Seek la cui sintassi è:

```
vecchiapos = Seek(file,posizione,modalità);
```

Il valore 'vecchiapos' ritornato indica la posizione assoluta (rispetto all'inizio del file) attuale, prima che l'operazione di spostamento abbia luogo; se il valore ritornato risulta -1, lo spostamento non è stato effettuato per qualche motivo (indice uscito fuori dai limiti del file); 'modalità' indica il tipo di riferimento da adottare per lo spostamento, che può valere OFFSET_BEGINNING (inizio del file), OFFSET_CURRENT (corrente posizione del file) o OFFSET_END (fine del file); 'posizione' è il valore che indica la nuova posizione da assumere rispetto al riferimento indicato da 'modalità' e può essere sia negativo che positivo, vediamo qualche esempio:

```
struct FileHandle *file;
LONG vecpos,pos;
  .
  .
vecpos = Seek(file,0,OFFSET_BEGINNING);
/* Posizionati all'inizio del file */
  .
vecpos = Seek(file,0,OFFSET_END);
/* Posizionati alla fine del file */
  .
vecpos = Seek(file,1,OFFSET_CURRENT);
/* Posizionati di un byte avanti alla posizione corrente */
  .
vecpos = Seek(file,-1,OFFSET_END);
/* Posizionati un byte prima della fine del file */
```

Fino ad adesso è stato sempre indicato come dopo ogni operazione DOS, sia possibile rilevare se quest'ultima è incorsa in un errore o meno, ma come fare per capire qual è l'errore che è stato causato? La funzione DOS che si occupa di identificare l'errore è IoErr:

```
(vp1)errore = IoErr();
```

(vp5)Dove errore è il codice dell'errore verificatosi; attenzione però se l'operazione ha avuto successo e quindi non si è verificato alcun errore, nulla si sà sul valore ritornato.
Ci sono molte altre funzioni della dos.library alcuni delle quali importanti, che corrispondono a nuove caratteristiche del 2.0 (assegnamenti multipli ecc.), ma quelle esaminate sono quelle che utilizzerete più spesso.

1.6 Lista errori DOS

Codici d'errore ritornati da IoErr().

ERROR_NO_FREE_STORE	103
ERROR_TASK_TABLE_FULL	105
ERROR_BAD_TEMPLATE	114
ERROR_BAD_NUMBER	115
ERROR_REQUIRED_ARG_MISSING	116
ERROR_KEY_NEEDS_ARG	117
ERROR_TOO_MANY_ARGS	118
ERROR_UNMATCHED_QUOTES	119
ERROR_LINE_TOO_LONG	120
ERROR_FILE_NOT_OBJECT	121
ERROR_INVALID_RESIDENT_LIBRARY	122
ERROR_NO_DEFAULT_DIR	201
ERROR_OBJECT_IN_USE	202
ERROR_OBJECT_EXISTS	203
ERROR_DIR_NOT_FOUND	204
ERROR_OBJECT_NOT_FOUND	205
ERROR_BAD_STREAM_NAME	206
ERROR_OBJECT_TOO_LARGE	207
ERROR_ACTION_NOT_KNOWN	209
ERROR_INVALID_COMPONENT_NAME	210
ERROR_INVALID_LOCK	211
ERROR_OBJECT_WRONG_TYPE	212
ERROR_DISK_NOT_VALIDATED	213
ERROR_DISK_WRITE_PROTECTED	214
ERROR_RENAME_ACROSS_DEVICES	215
ERROR_DIRECTORY_NOT_EMPTY	216
ERROR_TOO_MANY_LEVELS	217
ERROR_DEVICE_NOT_MOUNTED	218
ERROR_SEEK_ERROR	219
ERROR_COMMENT_TOO_BIG	220
ERROR_DISK_FULL	221
ERROR_DELETE_PROTECTED	222
ERROR_WRITE_PROTECTED	223
ERROR_READ_PROTECTED	224
ERROR_NOT_A_DOS_DISK	225
ERROR_NO_DISK	226
ERROR_NO_MORE_ENTRIES	232
ERROR_IS_SOFT_LINK	233
ERROR_OBJECT_LINKED	234
ERROR_BAD_HUNK	235
ERROR_NOT_IMPLEMENTED	236
ERROR_RECORD_NOT_LOCKED	240
ERROR_LOCK_COLLISION	241
ERROR_LOCK_TIMEOUT	242

```
ERROR_UNLOCK_ERROR                243
```

1.7 Exec

```
Exec

Exec stà per System Executive vale a dire Sistema Esecutivo che, come
si può intuire dal nome si occupa di gestire completamentamente la
macchina; l'exec.library mette a disposizione una serie di funzioni per
sfruttare l'exec (creazione di task, interrupt, comunicazioni ecc.); dato
che queste operazioni sono molto importanti e che verranno utilizzate molto
spesso, è giusto che vengano analizzate quì, prima di entrare nel vivo
della programmazione su Amiga.
La prima caratteristica da analizzare è
l'allocazione dinamica della memoria; dato che l'Amiga è un sistema
Multitasking e che tutte le strutture gestite sono dinamiche (cioè possono
venire allocate e utilizzate in un qualsiasi momento) non si può pretendere
che i programmi o i dati vengano a trovarsi in indirizzo fissi di memoria
(dato che in quello spazio potrebbe essere presente un qualsiasi altro
dato, creato da un altro programma), per cui ogni qualvolta che necessita
utilizzare memoria occorre chiederla al sistema operativo il quale, tramite
una lista interna provvederà ad allocarne la quantità richiesta e a
fornirne l'indirizzo. Vi è un altro modo per allocare spazi di memoria da
utilizzare nei programmi: tramite la dichiarazione delle variabili del C;
si badi bene però che questo corrisponde a spazio fisico occupato dal
codice, per cui se si dichiara un vettore di 250K, l'eseguibile sarà
allungato di appunto 250K introducendo così, un'inutile spreco di spazio su
supporto magnetico. La funzione dell'exec.library che si occupa
dell'allocazione di memoria è AllocMem ed ha la seguente sintassi:

BloccoMem = AllocMem(NumBytes,attributi);

Dove 'NumBytes' indica la lunghezza in bytes del blocco di memoria;
'attributi' indicano che tipo di memoria allocare e come deve essere
allocata; 'attributi' può valere:
MEMF_CHIP, indica che il tipo di memoria deve essere CHIP
MEMF_FAST, la memoria deve essere FAST
MEMF_ANY, la memoria può essere di qualsiasi tipo, in tal caso avrà
          precedenza la FAST, se questa non fosse disponibile si provvederà
          per la CHIP
MEMF_PUBLIC, questo flag indica che la memoria è di tipo pubblico può cioè,
             essere condivisa fra più task (questa caratteristica non è
             ancora implementata, il flag è stato inserito per usi futuri)
MEMF_CLEAR, utilizzando questo flag si cancella tutto il blocco di memoria
            prima che venga messo a disposizione del programma
MEMF_24BITDMA, (dalla verione 37 del S.O.) indica che il blocco di memoria
               verrà allocato nei primi 24 bit di indirizzo; questo poiché
               alcune schede Zorro II (che supporta solo 24 bit indirizzo
               perché creato con A2000 che sfruttava tali bit a causa del
               68000) potrebbero fare uso di memoria e quindi non possono
               "vedere" memoria al di fuori dei succitati 24 bit.
MEMF_REVERSE (V37), dato che nella ricerca del blocco di memoria richiesto
                    si procede a partire dagli indirizzi più bassi,
                    attivando questo flag si procede dall'indirizzo più alto
                    (le applicazioni che si osservano sovente sono i
```

caricatori dei S.O. in RAM che selezionano l'indirizzo più alto)

vi sono altri flag meno importanti per i quali si rimanda ai doc delle funzioni; 'BloccoMem' è il puntatore al blocco di memoria allocato ritornato dalla funzione, o NULL se non è stato possibile allocare il blocco, per cui ricordatevi di controllare sempre il valore restituito da tale funzione; vi assicuro che scrivere dati all'indirizzo 0 che corrisponde a NULL, porta probabilmente al cosiddetto GURU pittoresco riconoscibile non dalla solita scritta lampeggiante, ma da una serie di effetti video colorati (un Amiga si deve riconoscere anche quando si blocca); esempi:

```
UBYTE *buffer;
   .
   .
if ((buffer = (UBYTE *)AllocMem(10000,MEMF_CLEAR)) == NULL)
  printf("Errore non posso allocare buffer\n");
/* Alloca 10000 bytes di qualsiasi tipo e cancellali */
   .
if ((buffer = (UBYTE *)AllocMem(5000,MEMF_CHIP|MEMF_CLEAR)) == NULL)
  printf("Errore non posso allocare buffer\n");
/* Alloca 5000 bytes di memoria chip con cancellazione */
```

In entrambi gli esempi se non è possibile allocare i buffer verrà segnalato errore; ovviamente non bisogna indicare contemporaneamente flag in conflitto come MEMF_CHIP e MEMF_FAST. E qui si reincontra la prima regola dell'Amiga: "tutto ciò che viene allocato dovrà essere deallocato" per cui eccovi la funzione di exec che vi permette di resituire al sistema il blocco di memoria allocato, una volta che non ne avete più bisogno:

```
FreeMem(BloccoMem,NumBytes);
```

'BloccoMem' è il puntatore al blocco restituito da AllocMem e NumBytes la lunghezza di quest'ultimo in bytes; attenzione a non fornire un numero diverso da quello utilizzato per l'allocazione, non fornire un puntatore errato o non tentare di deallocare due volte lo stesso buffer, altrimenti andrete incontro ad un guru; esempio:

```
FreeMem(buffer,10000);
/* Deallocazione del buffer del primo esempio */
```

Dalla V36 del s.o. è disponibile una funzione di allocazione e una di deallocazione della memoria che ricorda la lunghezza del blocco allocato, per cui non vi è bisogno di indicarlo nella liberazione di quest'ultimo; il funzionamento di queste funzioni è identico alle corrispettive già esaminate:

```
BloccoMem = AllocVec(NumBytes,attributi);

FreeVec(BloccoMem);
```

Non tentate di utilizzare AllocMem con FreeVec o viceversa riguardo ad

AllocVec una FreeVec. Vi sono altre funzioni per la gestione della memoria come conoscere la quantità disponibile o altre che potrete trovare nei doc.

1.8 Liste e nodi

```
                Liste e nodi

Il sistema operativo mantiene durante il suo periodo di attività un elevato
numero di liste di dati di vario tipo: schermi, finestre, blocchi memoria,
task, interrupt ecc. Dato che tutte queste liste hanno un numero di elementi
variabile, vengono realizzate mediante strutture linkate; ciò significa che
oltre ai dati di un elemento della lista viene inserito un indirizzo che
costituisce il puntatore al prossimo elemento in modo da costituire una
catena. Exec provvede fornendo un implementazione standard per tutte le
liste che gestisce, mettendo a disposizione quindi le routine per ricerca,
inserimento e altre che non gravano così sul programmatore; la struttura
linkata realizzata con exec viene denominata coda (in inglese queue)
bidirezionale, cioè ogni elemento oltre ad avere il puntatore all'elemento
successivo ha anche il puntatore a quello precedente; in più vi sono due
elementi particolari, Head node e Tail node che puntano rispettivamente al
primo e all'ultimo elemento della lista e necessitano per iniziare una
qualsiasi procedura di ricerca. In termini pratici tutto ciò viene tradotto
tramite una struttura:

struct Node
{
  struct Node *ln_Succ;
  struct
  Node *ln_Prec;
  UBYTE ln_Type;
  BYTE ln_Pri;
  char *ln_Name;
};

Come vedete la struttura Node non solo è costituita dai puntatori al
nodo successivo e precedente ma anche da altri dati: 'ln_Type' indica il

                tipo
                 del nodo ed assume diverse definizioni a seconda dell'ambiente in ↩
                     cui
viene utilizzato che specificheremo volta per volta; 'ln_Pri' indica la
priorità dell'elemento (nella lista di task indica chi ha la precedenza
sugli altri) e per questo molte volte influisce sull'ordinamento della
lista (vale a dire viene prima l'elemento con priorità più alta) di solito
'ln_Pri' assume 0; 'ln_Name' è il puntatore ad una stringa che identifica
univocamente l'elemento e può essere utilizzato per la ricerca; questi tre
elementi sono opzionali e vedremo volta per volta quando vengono
utilizzati. Questa struttura deve essere definita all'inizio dell'elemento
dati della lista; ad esempio:

struct Persona
{
  struct Node LinkNode;
  char *Nome;
  char *Cognome;
  int anni;
  char *codicefis;
};
```

E’ possibile comunque utilizzare una struttura "minima" e cioè senza ln_Type, ln_Pri, ln_Name nella quale non fossero necessarie, denominata MinNode:

```
struct MinNode
{
  struct MinNode *min_Succ;
  struct MinNode *min_Pred;
};
```

Occorre mantenere anche le informazioni riguardanti l’indirizzo del nodo di testa e di quello di coda, per cui si utilizzerà la struttura List:

```
struct List
{
  struct Node *lh_Head;
  struct Node *lh_Tail;
  struct Node *lh_TailPred;
  UBYTE lh_Type;
  UBYTE lh_Pad;
};
```

lh_Type è un codice numerico che indica il tipo di lista (o meglio qual è il contenuto di codesta); lh_Pad non contiene alcun informazione e serve solamente ad assicurare l’allineamento a word di dati eventualmente presenti dopo; i primi tre valori sono puntatori a nodi e rappresentano la testa e la coda della lista; tale modo di conservare i puntatori non è quello convenzionale in quanto normalmente, si considerano i puntatori agli elementi di testa e di coda e si devono gestire una serie di casi particolari (inserimento in testa, inserimento in una lista vuota ecc.) implementando così una serie di verifiche di sicurezza; in questa struttura invece sono direttamente presenti gli elementi di testa e di coda da considerare "immaginari" perché non contengono alcun dato; in termini pratici lh_Head ed lh_Tail rappresentano rispettivamente il puntatore all’elemento successivo e all’elemento precedente di questo nodo di testa immaginario, ed il primo è il puntatore al primo nodo effettivo della struttura dati, ed il secondo vale 0; mentre lh_Tail e lh_TailPred costituiscono il successivo e il precedente del nodo di coda immaginario e valgono 0 per il primo e l’indirizzo dell’ultimo nodo effettivo per il secondo (lh_Tail è condiviso perché vale sempre 0); capite che facendo così si eliminano tutti i casi particolari e con essi le verifiche necessarie per implementarli; infatti in caso di lista vuota esistono comunque due elementi (la testa e la coda immaginari), per cui l’inserimento in una lista vuota è uguale all’inserimento normale, oppure l’inserimento di un nodo in testa alla lista equivale realmente ad un inserimento qualunque, poiché il nodo di testa della lista si trova sempre dopo quello immaginario è quindi non è realmente in testa anche se al programmatore sembra che sia così.
Dopo questa spiegazione da cane che si morde la coda, per inizializzare una lista e gestirla vi sono comunque delle funzioni di exec apposite e che quindi non richiedono nessuna operazione aggiuntiva da parte del programmatore.

```
struct MinList
{
  struct MinNode *mlh_Head;
  struct MinNode *mlh_Tail;
  struct MinNode *mlh_TailPred;
```

```
};
```

questa struttura viene utilizzata nella stessa maniera di List salvo che per questa non vanno specificate le informazioni aggiuntive. Osserviamo ora le funzioni messe a disposizione da Exec per la gestione delle liste:

```
NewList(Lista);
```

dove Lista è il puntatore ad una struttura List; NewList effettua l'inizializzazione della lista vuota ed imposta correttamente i valori di Lista.

```
AddHead(Lista,Nodo);
AddTail(Lista,Nodo);
Enqueue(Lista,Nodo);
```

dove Lista è sempre il puntatore ad una struttura List (se è appena creata occorre inizializzarla con NewList) e Nodo è il puntatore alla struttura Node da inserire nella lista; AddHead serve per inserire il nodo in testa alla lista, AddTail per inserirlo in fondo ed Enqueue per inserire il nodo in modo da mantenere la lista ordinata per priorità (ln_Pri); tenete presente che Enqueue deve operare su una lista già ordinata (vale a dire ogni elemento della lista deve essere stato inserito con questa funzione, oppure facendo attenzione che la posizione a seconda della priorità sia giusta); con Enqueue, il nodo verrà inserito dopo l'ultimo elemento con priorità maggiore o uguale a quella sua ed inoltre, il primo elemento della lista (quello di testa) ha priorità maggiore (quindi ordinamento numerico decrescente); questa funzione non può essere utilizzata ovviamente con liste minime.

```
Insert(Lista,Nodo,NodoPrec);
```

dove Lista e Nodo hanno lo stesso significato di prima e NodoPrec è il puntatore ad un nodo della lista "Lista"; Insert inserirà il nodo "Nodo" nella lista "Lista" nella posizione immediatamente successiva a NodoPrec.

```
Remove(Nodo);
Nodo = (struct Node *)RemHead(Lista);
Nodo = (struct Node *)RemTail(Lista);
```

Remove rimuoverà il nodo "Nodo" dalla lista in cui è presente; quì non occorre specificare nessuna lista giacché in Nodo stesso sono presente tutte le informazioni per la sua eliminazione (puntatore al nodo precedente ad a quello successivo); RemHead e RemTail rimuovono rispettivamente il nodo di testa e quello di coda e ne ritornano l'indirizzo in caso serva (per la sua

```
Nodo = (struct Node *)FindName(Lista,nome);
```

dove nome è il puntatore ad una stringa; FindName ricercherà il nodo con nome (ln_Name) "nome" nella lista "Lista" e se lo troverà ritornerà il suo indirizzo altrimenti ritornerà NULL; questa funzione non può essere utilizzata con le liste minime.

1.9 Lista dei tipi di nodi

```
Eccovi i diversi tipi di nodi specificati nel file di inclusione
<exec/nodes.h>:
NT_UNKNOWN      0L
NT_TASK         1L
NT_INTERRUPT    2L
NT_DEVICE       3L
NT_MSGPORT      4L
NT_MESSAGE      5L
NT_FREEMSG      6L
NT_REPLYMSG     7L
NT_RESOURCE     8L
NT_LIBRARY      9L
NT_MEMORY       10L
NT_SOFTINT      11L
NT_FONT         12L
NT_PROCESS      13L
NT_SEMAPHORE    14L
NT_SIGNALSEM    15L
NT_BOOTNODE     16L
```

1.10 I segnali (signals)

```
I segnali (signals)

I segnali (da non confondere con quelli di fumo che, se emessi dal vostro
calcolatore sono indice di qualcosa di grave) sono il mezzo a disposizione
di exec per "segnalare" al task che si è verificata una situazione per cui
quest'ultimo aveva chiesto di essere avvertito.
Questo meccanismo è molto importante per un sistema multitasking quale è
l'Amiga; infatti i processi (o task) vengono classificati con dei stati a
seconda del momento; in particolare dato che l'Amiga ha un solo processore,
questo deve essere diviso fra più task simulando più CPU virtuali; per far
questo viene dedicato ad ogni task un determinato quanto di tempo, alla
scadenza del quale il task viene momentaneamente congelato (con i contenuti
dei registri del microprocessore) per attivare un altro task (ciò che viene
definito multitasking pre-emptive); come potete osservare il task può
venire a trovarsi in diverse situazioni e più precisamente 3: attivo
(running), pronto (ready), addormentato (sleeping). Il task è in stato di
attivo quando è effettivamente in esecuzione, in stato pronto quando
attende di essere attivato dato che il microprocessore sta eseguendo un
altro task, ma quello che ci interessa al momento è lo stato di
addormentato; capita sovente infatti, che il programma debba attendere
senza far nulla un evento di I/O (pressione del taso del mouse o della
tastiera ecc.), per cui il tempo messo a sua disposizione verrebbe
totalmente sprecato dato che quest'ultimo potrebbe essere utilizzato da
altri task, ed il s.o. gli ridarebbe il controllo quando si verifica
l'evento di I/O atteso, ed è proprio a questo che serve lo stato di
addormentato; infatti il programma può chiedere di essere momentaneamente
"addormentato" tramite la funzione Wait di exec e verrà risvegliato solo
quando verrà raggiunto da un segnale (che potrà essere spedito dal s.o.
oppure da un altro task per un messaggio); un segnale è identificato da un
particolare bit in una maschera data da una LONG; il ché significa che
```

possono essere in circolazione al massimo 32 segnali per un task (di cui 16 riservati al s.o.), per cui bisogna innanzitutto chiederne l'uso al s.o. e liberarlo al più presto; in realtà non capita mai di utilizzare i segnali direttamente a meno di non voler segnalare un altro task (creato ovviamente dallo stesso programma) di effettuare una qualsiasi azione e comunque per la comunicazione fra task si utilizzano normalmente le porte ed i messaggi che fanno ugualmente uso di questo meccanismo; la funzione importante è ovviamente Wait che permette al programma di andare in stato di sleeping ottimizzando così le risorse di sistema; le funzioni per allocare/deallocare i segnali sono:

```
NumSegnale = AllocSignal(NumSegnale);
FreeSignal(NumSegnale);
```

dove NumSegnale è il numero di segnale da allocare (che va da 16 a 31 dato che i primi 16 sono occupati dal s.o.); AllocSignal alloca e riserva l'uso del segnale NumSegnale, il valore ritornato vale -1 se non è stato possibile allocare il segnale, oppure vale il codice del segnale allocato in caso contrario; FreeSignal rilascia l'uso del segnale precedentemente occupato, al sistema; c'è la possibiltà quando si alloca un segnale, e conviene utilizzarla, di non specificare un preciso segnale ma di farsi rilasciare il primo libero utilizzando come parametro di AllocSignal -1.
Ma la funzione di exec più importante riguardante i segnali, e che la ritroveremo più avanti, è la funzione Wait che permette di mandare il task in stato di sleeping in attesa di qualche segnale che lo svegli:

```
Segnali = Wait(SetSegnali);
```

dove SetSegnali indica per quali segnali il task deve essere riattivato e il valore ritornato Segnali (ambedue di tipo LONG) indica quale segnale ha risvegliato il task; SetSegnali è una maschera a cui ogni segnale di valore NumSegnale (ritornato da AllocSignal) corrisponde il bit di posizione n, dove n equivale a NumSegnale (se tale bit è ad 1 Wait attenderà per quel segnale altrimenti no); questo per permettere di specificare più segnali per il risveglio di un task, di conseguenza Segnali sarà dello stesso tipo esempio:

```
Segnali = Wait(1<<NumSegnale | 4);
if (Segnali & 1<<NumSegnale)
  printf("Task risvegliato dal segnale allocato.\n");
```

Il piccolo esempio pone il task in sleeping per il segnale NumSegnale allocato in precedenza e per il segnale 2 (4 = 1<<2) del s.o., quindi si verifica se il segnale che ha risvegliato il task è NumSegnale nel qual caso stampa un messaggio di conferma. Vi è anche la possibilità di verificare se un segnale è arrivato senza necessariamente mandare il task in sleeping tramite questa funzione di exec: SetSignal (vedere i doc per questo); l'ultima funzione da esaminare riguardo i segnali è Signal che permette al programma di inviare un qualsiasi segnale ad un task:

```
Signal(Task,SetSegnali);
```

dove Task è il puntatore ad una struttura Task che identifica il task a cui mandare il segnale (vedremo nella prossima puntata la struttura Task) e SetSegnali è la long tipo quella specificata in Wait contenente i bit dei segnali da inviare al task.

1.11 Le porte e i messaggi

Le porte e i messaggi

I messaggi e le porte sono il vero mezzo che permettono l'intercomunicazione fra task e s.o. (i segnali servivano solo per avvertire riguardo ad un particolare evento ma non permettono di scambiare dati); le porte vengono create dai task e permettono di ricevere il messaggio spedito (sono delle "porte" sul mondo di intercomunicazione), che altro non è che un blocco di dati il cui puntatore viene passato al task che riceve il messaggio e tramite il quale può acquisire tutte le informazioni che il mittente vuole inviargli; questo mezzo fa uso dei segnali (per avvertire il task che un messaggio è arrivato, segnale che comunque viene gestito dal s.o. che quindi non necessita di essere allocato o gestito dal programmatore) ed inoltre viene molto utilizzato da Intuition. Pensate a questa similitudine per comprendere bene come funziona tutto il meccanismo: supponete che voi (il task) dopo una nottata passata a lavorare davanti al vostro Amiga volete dormire un po' di più la mattina e quindi staccate il telefono, ponete la sveglia all'orario giusto (i segnali) e vi mettete a dormire (stato di sleeping); la mattina però arriva il postino che bussa due volte (tipico segnale del postino da non confondere con il vicino che anche se suona il campanello, si vuole dormire e si fa finta di niente), quindi vi svegliate (stato attivo) e prelevate il messaggio aprendo la porta. Vi sono comunque delle puntualizzazioni sui messaggi da considerare: il messaggio è una struttura dati composta da alcune informazioni necessarie al sistema e dal messaggio vero e proprio; il messaggio deve essere spedito ad una precisa porta, viene passato per indirizzo e deve essere risposto, cioè quando il programma ne riceve uno deve rispondere al mittente per far capire che il messaggio è stato ricevuto, inoltre quelli arrivati ad una porta vengono mantenuti in una coda in modo che nessuno di questi venga perso.
La porta come abbiamo detto è l'apertura sul mondo dei messaggi ed è identificata da questa struttura:

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

mp_Node serve perché tutte le porte vengono mantenute in una lista gestita dal s.o.;

mp_Flags
indica quale azione comporta l'arrivo di un messaggio a questa porta (vedere il riquadro); mp_SigBit è il numero di segnale utilizzato per segnalare al task che un messaggio è arrivato; mp_SigTask è il puntatore alla struttura task che indica il task a cui la porta è associata e che quindi deve essere segnalato (si possono creare più porte per un task); mp_SigTask può anche essere il puntatore ad una struttura Interrupt per la chiamata ad un interrupt software a seconda del valore di mp_Flags; mp_SigList è la lista dei messaggi arrivati alla porta che devono essere prelevati e risposti.
Per creare una porta si utilizza la funzione CreatePort:

```
Porta = (struct MsgPort *)CreatePort(nome,priorità);
```

dove "nome" è il puntatore ad una stringa che identifica univocamente la porta e "priorità" è la priorità che la porta avrà sulle altre (normalmente 0), il valore ritornato Porta è il puntatore alla struttura MsgPort creata da CreatePort; questa funzione assegna automaticamente la porta alla lista pubblica delle porte tramite il nome "nome" che permette di identificarla; vi è dalla V36 del s.o. una funzione equivalente:

```
Porta = CreateMsgPort();
```

questa funzione però non inserisce la porta nella lista di utilizzo pubblico; per poterla inserire nella lista pubblica (quindi creata con CreateMsgPort) occorre utilizzare AddPort, e per rimuoverla RemPort. La possibilità di inserire la porta nella lista pubblica ne permette l'utilizzo anche ad altri programmi che possono ricercare l'indirizzo della sua struttura (per potergli lanciare un messaggio) mediante questa funzione:

```
Porta = (struct MsgPort *)FindPort(nome);
```

FindPort insieme alla funzione per spedire il messaggio devono essere usate insieme (vale a dire, prima di lanciare un messaggio ricercare sempre la porta) e precedute da una chiamata Forbid(); (funzione di exec che inibisce l'attività degli altri task) e seguite da Permit(); (per ritornare alla situazione normale dopo Forbid), questo perché il task che ha creato la porta, potrebbe rimuoverla in un qualsiasi momento. Per rimuovere la porta creata dopo che non serve più si utilizzano le corrispettive funzioni di quelle utilizzate DeletePort e DeleteMsgPort con queste sintassi:

```
DeletePort(Porta);
DeleteMsgPrt(Porta); { V36 }
```

Illustriamo ora come i messaggi vengano inviati tramite le porte; innanzitutto, la struttura dati associata ad un messaggio è la seguente:

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

mn_Node serve perché i messaggi vengono mantenuti in una lista del s.o.; mn_ReplyPort è il puntatore alla porta che ha inviato il messaggio, a cui il programma deve rispondere dopo aver ricevuto quest'ultimo; mn_Length indica la lunghezza del messaggio compresa la struttura appena illustrata. Questa struttura indica solo le informazioni base per l'utilizzo del messaggio, ma il messaggio vero e proprio dove è, e che struttura ha? Il messaggio è situato nei bytes immediatamente successivi a quelli della struttura e possono avere qualsiasi forma vogliate e qualsiasi lunghezza (nei limiti imposti da mn_Length); per cui la forma di un ipotetico messaggio può essere questa:

```
struct Mio_Messaggio
{
  struct Message Mio_Msg;
  ULONG Mio_Dato1,_MioDato2;
```

```
  char Mio_Nome[20]; /* ecc... */
};
```

la struttura Message deve quindi essere sempre presente in testa ad ogni messaggio.
Per inviare un messaggio ad una porta occorre utilizzare la funzione di exec PutMsg:

```
PutMsg(porta,messaggio);
```

dove "porta" è il puntatore alla struttura MsgPort che identifica la porta a cui inviare il messaggio e "messaggio" è il puntatore alla struttura Message del messaggio da inviare; si consiglia di utilizzare questa funzione insieme a FindPort e di racchiuderle fra due chiamate Forbid() e Permit(), poiché il task della porta che riceve il messaggio, può decidere di chiuderla in un qualsiasi momento! Tramite la chiamata Forbid(), si impedisce il multitasking per cui il task non può fare nulla e tantomeno chiudere una porta; Permit() in ultimo riporta il sistema alla normalità. Qualche riga sopra è stato scritto si consiglia perché non è detto che sia sempre necessario comportarsi così; mettiamo infatti che il task proprietario della porta sia creato e messo in funzione dal vostro programma (che è un task a sua volta); in questo caso sapete benissimo a priori come questo si comporterà, e magari sarà il vostro programma a decidere quando fargli chiudere la porta, per cui il problema non si pone.
Per attendere un messaggio, dovreste averlo capito dalla scorsa puntata, occorre utilizzare la funzione Wait; il codice del segnale per cui Wait deve attendere è presente nella struttura MsgPort della porta ché riceverà il segnale; per cui il codice con l'istruzione Wait per attendere il messaggio di una porta "porta" può avere la seguente forma:

```
ULONG SegnPorta,segnali;
  .
  .
SegnPorta = 1 << porta->mp_SigBit;
segnali = Wait(SegnPorta);
```

Se però come nel caso appena descritto, l'evento per cui attendere è solo un messaggio, si può utilizzare una funzione che fa tuttò ciò automaticamente, passando semplicemente il puntatore alla porta per cui ricevere il messaggio:

```
messaggio = (struct Message *)WaitPort(porta);
```

"messaggio" è il puntatore alla struttura Message del messaggio arrivato alla porta; occorre ricordare che il sistema mantiene una lista FIFO (First In First Out, cioè il primo ad entrare è il primo ad uscire) di tutti i messaggi che arrivano alla porta per evitare che vadano persi; Wait e WaitPort non eliminano il messaggio dalla lista, per cui occorre utilizzare la funzione GetMsg() che ritorna anch'essa il puntatore del primo messaggio rimasto nella lista, eliminandolo però da quest'ultima (morale: "messaggio" ritornato da WaitPort non serve a nulla, poiché deve essere ripescato da GetMsg):

```
messaggio = (struct Message *)GetMsg(porta);
```

Come già accennato, bisogna rispondere ad ogni messaggio arrivato alla

propria porta per segnalare che il messaggio è stato ricevuto correttamente, mediante la funzione ReplyMsg:

```
ReplyMsg(messaggio);
```

dove "messaggio" è il puntatore al messaggio a cui bisogna rispondere.

1.12 Lista dei valori di mp_Flags

Valori di mp_Flags nella struttura MsgPort:

PA_SIGNAL:
se arriva un messaggio a questa porta allora avvisare il task mp_Task con il segnale mp_SigBit
PA_SOFTINT:
se arriva un messaggio alla porta allora chiamare una routine di interrupt software il cui puntatore alla relativa struttura Interrupt è presente in mp_Task
PA_IGNORE:
se arriva un messaggio non fare niente; questo serve per bloccare segnalazioni o chiamate ad interrupt senza modificare il campo mp_Task.

1.13 I semafori

I semafori

E ci riferiamo naturalmente non a quelli dell'incrocio sotto casa; questo è un argomento che tratteremo data la sua importanza teorica, e non vi capiterà quasi mai di utilizzare i semafori direttamente, ma sempre mediante apposite funzioni del s.o. che verranno illustrate di volta in volta.
In un sistema multitasking quale è l'Amiga, le risorse del sistema vengono condivise fra più task per cui bisogna far attenzione che, quando una risorsa viene utilizzata da un task, non deve essere accedibile da un altro; immaginate ad esempio cosa può succedere se, mentre un word-processor sta stampando un testo, un altro task che può essere un programma di grafica tenta di stampare anch'egli; in una tale evenienza i dati miscelandosi sulla parallela produrranno dei risultati catastrofici.
Per cui vengono implementate dal sistema delle funzioni di bloccaggio e sbloccaggio (lock e unlock) e sono utilizzate dal task che deve rispettivamente appropiarsi della risorsa per usarla e una volta finito, rilasciarla al sistema; mentre la risorsa è bloccata nessun altro può accedervi finché non viene liberata. Bisogna stare attenti nell'utilizzo di queste funzioni di lock e unlock perché si entra facilmente nelle situazioni di stallo dette in inglese dead-lock; qualche tempo fa capitò proprio una situazione di questo tipo con una risorsa del sistema che tutti conoscete benissimo: il Blitter (ebbene sì lo ammetto, anche io sbaglio); in questa situazione ci si appropriava della risorsa mediante la funzione di bloccaggio relativa (OwnBlitter che vedremo in una delle prossime puntate) e prima di rilasciarla si utilizzava la funzione Text della graphics.library (scrive un testo specificato) che utilizza il Blitter per la scrittura del testo; il programma si bloccava, poiché per andare avanti attendeva il ritorno della funzione Text, ma quest'ultima allo stesso tempo

attendeva che il Blitter venisse liberato.
Il metodo per gestire le risorse è basato su questo principio ma è molto più efficiente: i semafori; infatti con il semplice bloccaggio della risorsa, non si assicura che tutti la possano utilizzare equamente, poiché se un processo si accorge che la risorsa è occupata e preferisce fare qualche

controllare prima di lui e occuparla anche se ne aveva diritto il task di prima. I semafori non sono altro che liste che indicano quale sarà il task che ha diritto al prossimo utilizzo della risorsa; per cui la funzione di bloccaggio (come OwnBlitter che opera proprio in questo modo) se la risorsa non è libera, inserirà il task nella lista del semaforo e ne restituerà il controllo quando il processo ha effettivamente diritto di utilizzo.

1.14 I processi (task)

I processi (task)

Un task (o processo in italiano) dovrebbe essere oramai chiaro a tutti cosa sia, infatti altro non è che un programma (in inglese vuol dire lavoro o mansione); dato che il s.o. dell'Amiga è multitasking appare ovvio, che più task possono essere attivi contemporeanemente e quindi risulta logico che il programma principale possa attivare un secondo task che procede parallelamente. Un task è identificato da una struttura denominata "Task" che mantiene tutte le informazioni relative alla creazione ed alla vita del processo; è possibile ottenere il puntatore a tale struttura del task in esecuzione mediante la chiamata:

```
processo = FindTask(NULL);
```

il task in esecuzione di cui verrà ritornato il valore "processo" puntatore alla struttura Task, è sicuramente quello che esegue la chiamata alla funzione appena vista; se si vuole ricerca un task in particolare, bisogna invece specificare la stringa indicante il nome che lo identifica nella lista.
Per la creazione di un processo occorre allocare la memoria della struttura Task mediante AllocMem con attributi MEMF_CLEAR e MEM_PUBLIC; in tal modo la struttura verrà inizializzata con zero e sarà di tipo condivisibile; i campi da inizializzare della struttura Task dipendono da come intendi utilizzare il processo; nel caso più semplice occorre inizializzare i seguenti campi:
tc_Node - dato che il processo verrà inserito nella lista dei task del sistema occorre inizializzare il nodo con le informazioni relative alla priorità, il tipo ed il nome.
tc_SPLower - il limite di memoria inferiore dello stack del task
tc_SPUpper - il limite di memoria superiore dello stack
tc_SPReg - il valore iniziale dello stack pointer (SP)
bisogna porre a zero tutti gli altri campi (ma allocando la memoria della struttura con MEMF_CLEAR viene già fatto).
Una volta che la struttura Task è inizializzata il processo può essere reso attivo con AddTask:

```
AddTask(processo,PCiniziale,PCfinale);
```

dove "processo" è il puntatore alla struttura Task contenente i dati relativi alla creazione del task; PCiniziale è l'indirizzo della prima

istruzione del task da eseguire e PCfinale è l'indirizzo della procedura che verrà eseguita all'uscita del task (con PCfinale uguale a NULL ne viene implementata una standard di exec); dalla versione 2 del s.o. AddTask ritorna l'indirizzo del task o NULL in caso in cui l'operazione ha fallito. Per rimuovere il task (che potrebbe essere già stato rimosso nel caso la sua esecuzione sia giunta al termine) occorre utilizzare RemTask:

```
RemTask(processo);
```

Lo stesso lavoro per la creazione di un task può essere effettuata con una singola chiamata a CreateTask (una macro di amiga.lib):

```
processo = (struct Task *)CreateTask(nome,priorità,PCiniziale,grandstack);
```

dove "nome" è il nome che avrà il task nella lista, "priorità" è la sua priorità, "PCiniziale" l'indirizzo della prima istruzione del task e "grandstack" la grandezza in bytes dello stack per il task; occorre far attenzione alla grandezza dello stack (che con AddTask deve essere allocato dal programma, mentre con CreateTask è fatto automaticamente) perché questo non solo dipende dalla quantità di dati che vengono posti sullo stack dal suo task, ma anche dal s.o., poiché nel task-switching i registri del task congelato vengono memorizzati proprio nel suo stack per essere poi ripresi quando verrà riattivato (si consiglia un minimo di 256 bytes); per "eliminare" un task creato con CreateTask prima del suo tempo occorre eseguire DeleteTask:

```
DeleteTask(processo);
```

Fare attenzione che prima che un task venga eliminato, tutte le risorse allocate da quest'ultimo (memoria, librerie ecc.) devono essere liberate; per maggior sicurezza conviene che, tutte le risorse che necessitano al task figlio (quello da creare) vengano allocate dal task padre (il processo che crea il task figlio).
Abbiamo ultimato l'argomento dei task anche se vi sono altre importanti caratteristiche come il trattamento dei traps del task che vedremo eventualmente di trattare in un articolo a parte.

1.15 Il server di interrupt

Il server di interrupt

Alla chiamata del server di interrupt vengono passati dal sistema una serie di importanti informazioni dal sistema in particolari registri (per questo si consiglia di scrivere il codice delle interrupt in assembler). I parametri dell'interrupt hardware diretta (che viene installata con SetIntServer) sono:

D0 - scratch (viene utilizzato dal sistema quindi il suo valore verrà mutato)
D1 - contiene i valori di INTENAR (il registro che indica quali interrupt sono abilitate) e INTREQR (il registro che indica quali interrupt si sono verificati) "and"ati fra loro, ed indica quali interrupt si sono verificate effettivamente.
A0 - inidirizzo base del set di chip custom.
A1 - valore presente nel campo di is_Data dell'interrupt (quindi il

```
     puntatore alle informazioni che l'installatore dell'interrupt vuole
     passare)
A5 - è usato come un vettore al tuo codice di interrupt

     exec
Un codice di interrupt diretto deve ritornare con RTS (non con RTE come si
dovrebbe fare per un installazione dell'interrupt diretto via hardware);
prima di uscire bisogna cancellare il bit dell'interrupt relativo nel
registro INTREQ per indicare che l'interrupt è stata servita.

I parametri dell'interrupt hardware installata con AddIntVector:
D0 - scratch
D1 - scratch
A0 - scratch (tranne alcuni casi)
A1 - valore del campo is_Data nella struttura Interrupt
A5 - puntatore al codice di interrupt (scratch)
A6 - scratch
in questo caso dato che l'interrupt non è direttamente connessa al sistema
non vi è bisogno di resettare il bit nel registro INTREQ; vi è però la
possibilità di saltare tutte le interrupt che vengono dopo nella lista dei
servers ponendo il flag Z a 1 (cancellato); nel caso in cui vogliate far
seguire l'esecuzione a tutte le interrupt della lista bisogna settare il
flag Z (porre a 0); il modo migliore per pilotare il flag Z è questo:

Setzflag_Chiamalaprossima:
  MOVEQ #0,D0
  RTS

Clearzflag_finiscilacatena:
  MOVEQ #1,D0
  RTS

I server di interrupt di questo tipo (installati con AddIntServer) possono
essere attivati solo per le seguenti interrupt: PORTS, COPER, VERTB, EXTER e
NMI.
```

1.16 Interrupt

```
                Interrupt

L'interrupt (in italiano interruzione) è un segnale che viene inviato
dall'hardware (quasi sempre adibito a mansioni di I/O) al microprocessore
che interrompe per questo il normale corso di elaborazione del programma per
andare ad eseguire una particolare routine (detta interrupt server o
servitore dell'interruzione) che risolverà la particolare situazione del
sistema; ad esempio un classico caso di interrupt è la tastiera: appena
l'utente preme un tasto, l'interfaccia della tastiera invia un interrupt al
microprocessore, il quale esegue la routine di interruzione che preleva il
codice del tasto premuto; in questa maniera si evita che il microprocessore
verfichi sempre se un tasto è premuto (polling) risparmiando così, un
notevole numero di cicli macchina; potete osservare la lista delle diverse
interruzioni nella
                tabella
                 apposita.
Accanto a questo tipo di interrupt denominati hardware vi sono i software
```

interrupts, cioè interrupt provocati dal software (dal task quindi); voi direte: a questo punto conviene creare un subroutine da chiamare quando serve e si ottiene lo stesso risultato! Invece non è proprio la stessa cosa, poiché le interrupt software hanno una priorità più alta dei task (e più bassa delle interrupt hardware) quindi eseguendo un interrupt software il multitasking è momentaneamente disattivato.
La struttura che definisce un'interrupt sia software che hardware è la seguente:

```
struct Interrupt
{
  struct Node is_Node;
  APTR id_Data;
  VOID (*is_Code)();
};
```

dove is_Node indica che l'interrupt è mantenuta in una lista del s.o. e devono essere definite le sue caratteristiche, priorità, nome e tipo (NT_INTERRUPT); is_Data è un puntatore ad un buffer dati gestito dal programmatore per passare all'interrupt, dati che possono essere utili; is_Code è il puntatore alla routine di interrupt.
Le funzioni di exec che servono per l'installazione di interrupt sono:

```
vecchiaint = (struct Interrupt *)SetIntVector(tipoint,nuovaint);
```

dove "tipoint" è una costante definita in <hardware/intbits.h> che indica a quale interrupt hardware installare il server identificato dal puntatore alla struttura Interrupt "nuovaint"; "vecchiaint" è il puntatore alla struttura Interrupt del server che viene rimpiazzato; questa funzione esclude il codice di interrupt esistente, e questo potrebbe causare qualche problema al sistema a cui potrebbe servire l'interrupt, quindi va usata con cautela.

```
AddIntServer(tipoint,nuovaint);
```

Questa funzione invece aggiunge l'interrupt a quelle già esistenti; praticamente quando si verifica l'interrupt di tipo "tipoint" il sistema scandirà una lista (ordinata per priorità) con i diversi interrupt servers e li eseguirà tutti; questo metodo quindi non mette in pericolo il perfetto funzionamento del sistema; per eliminare l'interrupt dalla lista dei servers occorre chiamare RemIntServer con gli stessi parametri della AddIntServer.
La funzione per chiamare un'interrupt software è questa:

```
Cause(inter);
```

dove "inter" è il puntatore ad una struttura Interrupt. In realtà come abbiamo visto la puntata scorsa l'interrupt software può essere chiamata da una porta appena giunge un messaggio ponendo in mp_Flags PA_SOFTINT

```
                e in
mp_SigTask il puntatore alla struttura Interrupt relativa.
```

1.17 Lista delle interrupt

```
Lista delle interrupt

Lista delle diverse interrupt hardware presenti nell'Amiga; la priorità
hardware indica l'effettiva priorità delle interruzioni (il 680x0 ha 8
livelli di interrupt); queste interruzioni vengono ulteriormente suddivise
(e quindi utilizzate in più casi) per cui viene specificata una "pseudo"
priorità di exec:

Priorità  Priorità  Descrizione                                Etichetta
Hardware    Exec

             1      Buffer di trasmissione seriale vuoto     TBE
   1         2      Completato blocco disco                  DSKBLK
             3      Interrupt software                       SOFTINT
   2         4      CIAA e interrupt esterna INT2            PORTS
             5      Coprocessore grafico (COPPER)            COPER
   3         6      Intervallo di vertical blank             VERTB
             7      Blitter ha finito                        BLIT
             8      Canale audio 2                           AUD2
   4         9      Canale audio 0                           AUD0
             10     Canale audio 3                           AUD3
             11     Canale audio 1                           AUD1
   5         12     Buffer di ricezione seriale pieno        RBF
             13     Pattern di sincronismo disco trovato     DSKSYNC
   6         14     CIAB e interrupt esterna INT6            EXTER
             15     Special (abilitatore globale)            INTEN
   7         -      Interrupt non mascherabile               NMI
```

1.18 Le librerie (library)

```
                Le librerie (library)

Arriviamo ad uno degli argomenti più importanti di questa puntata (l'ultimo
di Exec) che permetterà di aprire gli altri più concreti.
In "introduzione alle librerie", avevamo già accennato che cosa è una
libreria e avevamo detto che un insieme di funzioni che permettono
all'utente di pilotare correttamente il sistema in tutte le sue parti,
dall'apertura di un file a quella di uno schermo. La libreria è composta da
una struttura comprendente le informazioni di quest'ultima e una tabella di
salti per le funzioni vere e proprie; la libreria, per poter utilizzare le
sue funzioni, deve essere aperta mediante la chiamata:

libreria = (struct Library *)OpenLibrary(nome,versione);

dove " LINK  è il puntatore alla struttura Library della libreria e viene
utilizzata per ogni chiamata a sue funzioni; "nome" è il puntatore ad una
stringa contenente il nome della libreria e "versione" è una long che indica
```

la versione della libreria (che corrisponde a quella del sistema esempio 32, 33, 36 ecc.).
Qui occorre ricordare la famosa regola di Amiga: "quello che viene aperto deve essere chiuso"; per cui eccovi la funzione per chiudere la libreria prima dell'uscita dal vostro programma:

```
CloseLibrary(libreria);
```

dove

libreria
è il puntatore alla struttura Library ritornato da OpenLibrary.

Il valore "libreria" non verrà mai utilizzato se non nella chiusura, perché nella chiamata alle funzioni di libreria, i parametri vengono passati per registri, mentre in una qualsiasi chiamata ad una funzione da C i parametri sono passati per stack; in realtà quindi, vengono chiamate delle funzioni presenti nella libreria del compilatore (solitamente Amiga.lib) che a loro volta passeranno i parametri dallo stack nei registri oppurtuni e qui chiameranno la funzione vera e propria mediante il puntatore "libreria"; perciò i puntatori alle strutture Library delle librerie da utilizzare vanno dichiarati dal programmatori come variabili esterne (o globali) con un nome ben preciso che non può essere mutato e che potete osservare nella

tabella
rispettiva.

1.19 Lista delle librerie

Lista delle librerie

Elenco dei nomi di libreria con i relativi nomi dei puntatori alle strutture Library relative:

Nome libreria	Nome puntatore base della libreria
asl.library	AslBase
commodities.library	CxBase
diskfont.library	DiskfontBase
dos.library (*)	DOSBase
exec.library (*)	SysBase
expansion.library	ExpansionBase
gadtools.library	GadToolsBase
graphics.library	GfxBase
icon.library	IconBase
iffparse.library	IFFParseBase
intuition.library	IntuitionBase
keymap.library	KeyMapBase
layers.library	LayersBase
mathffp.library	MathBase
mathieeedoubbas.library	MathIeeeDoubBasBase
mathieeedoubtrans.library	MathIeeeDoubTransBase
mathieeesingbas.library	MathIeeeSingBasBase
mathieeesingtrans.library	MathIeeeSingTransBase
mathtrans.library	MathTransBase
rexxsys.library	RexxSysBase
rexxsupport.library	RexxSupBase

```
translator.library              TranslatorBase
utility.library                 UtilityBase
version.library                 (uso privato per il sistema)
workbench.library               WorkbenchBase

(*)Vengono aperte automaticamente dal modulo di partenza del codice in
linguaggio C.
```

1.20 Versioni del s.o.

```
Corrispondenza tra la versione del s.o. e la versione numerica:
V32     v1.2
V33     v1.3
V34     v1.4
V36     v2.0
V37     v2.04
V39     v3.0
V40     v3.1
```

1.21 L’interfaccia grafica di Amiga e Intuition

```
L’interfaccia grafica di Amiga e Intuition

Una delle caratteristiche per cui l’Amiga è diventato famoso, riguarda
la sua interfaccia grafica user-friendly; con interfaccia si intende
normalmente il sistema tramite il quale, l’utente scambia informazioni con
il sistema operativo, e quella dell’Amiga è una tipica GUI (Graphical User
Interface ossia Interfaccia Utente Grafica) vale a dire pesantemente basata
sulla grafica; infatti l’utente pilota l’Amiga mediante icone, menù,
finestre ecc., vale a dire un sistema "intuitivo" e da quì il nome di
intuition. Intuition quindi, è quella parte del sistema operativo
(comprendente anche intuition.library e altre librerie) che ha due
fondamentali obiettivi: il primo è quello di gestire l’interfaccia
amichevole tra il sistema e l’utente ed il secondo, ben più importante, è
quello di mettere a disposizione del programmatore delle librerie per
gestire questa interfaccia, in modo che anche i propri programmi possano
usufruire delle caratteristiche di intuition. Dal punto di vista
dell’applicazione intuition è formata dalle seguenti componenti: screen,
window, menu, gadget, requester e input event; per suddividere lo spazio
grafico messo a disposizione dal video si utilizzano screens (schermi) e
windows (finestre); uno schermo è uno spazio che occupa tutto il quadro
video e possiede delle proprie caratteristiche di risoluzione e numero di
colori; uno schermo può essere slittato su un altro (questo per merito del
copper) come dei fogli; si possono quindi inserire sullo schermo diverse
finestre (possiamo intenderle come delle "finestre" su diversi mondi, anche
se window è un termine originario di Unix) ognuna contenente diverse
informazioni; il perché di questo complicato metodo di suddivisione dello
spazio grafico è presto detto; lo schermo del workbench necessita
mediamente di 16 colori che sono più che sufficienti per visualizzare
finestre e icone (a meno che non abbiate inserito l’immagine di una bella
fanciulla sullo sfondo, nel qual caso ve ne necessitano almeno 256); però
se deve partire un programma di grafica pittorica sarebbe assurdo
```

visualizzarlo come finestre sullo schermo workbench; d’altra parte sarebbe uno spreco inizializzare lo schermo workbench con 256 colori prevedendone l’utilizzo con applicazioni grafiche (come succede sotto Windows); per questo l’idea di un ulteriore classificazione di spazio grafico mediante schermi è molto utile ed importante. I menù sono una lista di opzioni a "scomparsa", vale a dire non visibili normalmente sullo schermo, ma premendo il tasto destro del mouse compaiono, e permettono la selezione di una particolare opzione. I gadgets (o volendo tradurli bottoni) sono degli oggeti grafici che selezionati mediante il puntatore del mouse e la pressione del tasto sinistro compiono un operazione; ne esistono di diversi tipi come vedremo ed addirittura esiste BOOPSI un sottosistema di intuition che ne implementa l’uso e l’architettura mediane programmazione orientata agli oggetti. I requesters sono delle finestre che effettuano delle richieste all’utente. Input event è il metodo con cui Intuition comunica all’applicazione l’avvenimento di un evento di ingresso (selezione del menù, pressione di un gadget ecc.).

1.22 Gli schermi

Gli schermi

Prima di utilizzare una qualsiasi risorsa Intuition occorre aprire l’intuition.library dato che utilizzeremo le sue funzioni. Bene iniziamo col descrivere gli schermi; nelle vecchie versioni del chip set di Amiga esistevano solo 4 risoluzioni possibili (320x256, 320x512, 640x256, 640x512) di cui due (quelle di ampiezza di 320 pixels) potevano arrivare a 32 colori su una palette di 4096 e utilizzare particolari modalità come l’EHB (Extra Half Brite) e l’HAM (Hold And Modify) che permettevano rispettivamente, con alcune limitazioni, di raggiungere 64 e 4096 colori, mentre quelle con 640 pixels di ampiezza dovevano accontentarsi di un massimo di 16 colori. L’avvento dell’ECS ha portato solo qualche nuova risoluzione, mentre la vera rivoluzione si è avuta con l’AGA, data la sua possibile riprogrammazione delle frequenze video si possono ottenere una miriade di nuove risoluzioni, tutte con un massimo di 256 colori su una palette di 16.7 milioni di colori circa, e tutte con la possibilità di utilizzare la modalità HAM8, che permette di visualizzare con qualche limitazione tutta la palette di 16.7 milioni (attenzione però non si tratta di true-color, anzi in questa modalità lo schermo viene memorizzato sempre con 8 bitplanes). Come già accennato, possono esistere più schermi contemporaneamente sul vostro video, ed inoltre gli schermi possono essere di due tipi: public e custom; gli schermi custom vengono utilizzati solo dai programmi che li creano, per cui le finestre visualizzate su di essi possono essere create solo dall’applicazione che ha aperto gli schermi; gli schermi public sono "pubblici", vale a dire che possono essere utilizzati anche da altre applicazioni; quindi una finestra creata da un programma, in cui non viene specificato su quale schermo essere visualizzata può andare a finire su uno di questi schermi public; uno schermo public è il workbench e prima della versione 2.0 del sistema era l’unico possibile.
La struttura che identifica uno schermo e che ne contiene tutte le informazioni necessarie è la struttura

Screen

.

Le funzioni principali di intuition.library che riguardano gli schermi sono OpenScreenTags(), OpenScreenTagList(), OpenScreen() e CloseScreen(); le prime tre servono per la creazione di uno schermo, mentre l’ultima per la

sua chiusura e per liberare la memoria occupata da questo. Le prime due funzioni di apertura sono molto simili e sono state introdotte dalla versione 2.0 (V36) del sistema, perché usano un modo molto efficiente per il passaggio di parametri, denominato per "Tags" introdotto appunto nel 2.0 e che ritroveremo in moltre altre funzioni di libreria; la creazione dei Tags è avvenuta nel 2.0 perché in quel momento ci si è posti il problema del passaggio di parametri; infatti prima i parametri per l'apertura dello schermo, venivano passati mediante una struttura NewScreen contenente le diverse caratteristiche di quest'ultimo; dato il radicale cambiamento del sistema nel 2.0 molti parametri sono stati aggiunti e quindi, si è posto l'accento sulla compatibilità con versioni future del sistema (infatti il NewScreen non valeva più a molto, ed è stato mantenuto per compatibilità con il vecchio sistema), per cui è stato creato il Tag; il Tag è una struttura di due elementi, di cui il primo contiene un valore identificatore che indica quale parametro si vuole passare, ed il secondo rappresenta il parametro vero e proprio; in questa maniera si può passare alla funzione una lista di tags che contengono i diversi parametri e, dato che il numero di tags è variabile, la compatibilità con il futuro è assicurata; in più se qualche parametro (ad esempio una vecchia chiamata con una nuova funzione) non dovesse essere passato, il sistema provvederà ad utilizzare un valore standard per quella caratteristica. Come prima avevamo accennato esistono due funzioni per l'apertura dello schermo mediante tag, OpenScreenTags() e OpenScreenTagList(); la differenza consiste semplicemenete che, nella prima funzione i tags vengono passati come parametri della procedura, mentre nella seconda viene passato il puntatore ad un array di tags; dato che il numero di parametri con questo metodo è variabile, vi starete chiedendo come si fà ad indicare alla procedura che la lista è finita; per indicare che la lista è finita, basta semplicemente inserire come ultimo tag un codice speciale che indica appunto la fine della lista, ed è una costante di un file di inclusione:

TAG_END, oppure TAG_DONE

Dopo questa piccola parentesi, analizziamo le funzioni di apertura dello schermo:

```
schermo = OpenScreenTags(nuovoschermo,tagId1,tagVal1,TagId2,TagVal2,.....);
```

dove "schermo" è il puntatore alla struttura
Screen
identificante lo
schermo creato, o NULL se l'operazione è fallita; "nuovoschermo" è il puntatore alla struttura NewScreen (utilizzando i tag può essere tranquillamente impostato a NULL) e dopo seguono la lista di tag degli

attributi
dello schermo che devono ultimare, lo ricordiamo, con TAG_END o TAG_DONE.

```
schermo = OpenScreenTagList(nuovoschermo,listatags);
```

dove "listatags" è il puntatore ad un array di strutture TagItem; TagItem è la struttura contenente i due valori del "tag" ed ha la seguente forma:

```
struct TagItem
{
  ULONG ti_Tag; /* codice dell'attributo */
```

```
  ULONG ti_Data; /* valore dell'attributo */
};
```

potete vedere i diversi utilizzi di OpenScreenTags e OpenScreenTagList negli esempi, ma oramai la differenza dovrebbe essere chiara.

```
schermo = OpenScreen(nuovoschermo);
```

OpenScreen è la vecchia funzione di apertura degli schermi, utilizzata normalmente prima della versione 2.0 del sistema ed il passaggio dei parametri avviene mediante la struttura NewScreen; vi è comunque la possibilità di sfruttare le nuove qualità degli schermi del 2.0 utilizzando questa funzione, mediante la struttura ExtNewScreen; questa struttura è praticamente identica a NewScreen fuorché che possiede un campo in più, presente alla fine della struttura, ed è un puntatore all'array di tags degli attributi per lo schermo secondo la modalità vista; la struttura ExtNewScreen deve possedere il bit NS_EXTENDED settato nel campo Type.
In questa maniera se il programma gira con una versione del sistema operativo inferiore al 2.0 allora la struttura verrà interpretata come una comunissima NewScreen altrimenti, verrà considerato l'ultimo campo e verrano utilizzato gli attributi del 2.0; da notare che questa possibilita è presente dalla versione V37 del s.o.
Come ultima funzione rimane da analizzare CloseScreen:

```
CloseScreen(schermo);
```

CloseScreen come potete immaginare provvede a chiudere uno schermo precedentemente aperto.
Torniamo a parlare di schermi pubblici; per poter aprire uno schermo pubblico basta specificare, il tag

```
SA_PubName
```

con il nome pubblico da assegnare allo schermo, nella lista dei tag per la funzione OpenScreenTags; vi sono però molte altre funzioni di intuition che riguardano gli schermi pubblici e che servono per la loro gestione; se volete ad esempio utilizzare uno schermo "public", per aprire una finestra o altro, occorre avere il puntatore alla struttura Screen di quest'ultimo; la funzione che permette di ottenere il puntatore ad uno schermo pubblico con un determinato nome è LockPubScreen; questa funzione ha il compito anche di "bloccare" momentaneamente lo schermo in modo che nessun altro task o sistema ne possa modificare lo stato (chiudendolo ad esempio); in questa maniera si evita di aprire una finestra, proprio mentre il sistema chiude lo schermo (che porterebbe a risultati disastrosi, come potete ben immaginare).
Una volta che avete eseguito tutte le operazioni che volevate sullo schermo pubblico occorre sbloccarlo mediante la funzione UnlockPubScreen.
Come accennato la volta scorsa esiste uno schermo pubblico di default, che normalmente è il Workbench; lo schermo pubblico di default può però essere cambiato, mediante la funzione SetDeafultPubScreen, oppure si può conoscere quale sia mediante GetDefaultPubScreen.
Vi è talvolta la necessità di accedere a tutti i public screens, per avere una lista completa da utilizzare in diversi modi (visualizzandola all'utente per una selezione ad esempio); in questo caso occorre utilizzare la funzione LockPubScreenList che blocca la lista di sistema desgli schermi pubblici; il nodo della lista è descritto mediante la struttura PubScreenNode definita in "intuition/screens.h"; una volta ultimata l'analisi sulla lista degli schermi pubblici bisogna liberarla mediante UnlockPubScreenList.
Comunque non serve utilizzare direttamente le informazioni della lista, in

quanto basta richiamare la funzione NextPubScreen che restituisce il prossimo schermo da esaminare; quando l'ultimo schermo è stato restituito la funzione ritornerà NULL.

DrawInfo e 3D Look

Dalla versione 2.0 del sistema si adotta uno speciale gioco di colori, per creare effetti tridimensionali nelle finestre e nei gadget; questi effetti 3D sono ottenuti associando ad un colore una definizione del genere "bordo ombra" o "bordo luminoso", in modo che il sistema sappia a quale colore ricorrere se deve disegnare un bordo illuminato o in ombra; nell'apertura di uno schermo, possono essere specificate in
SA_Pens
, quali
penne
corrispondano
al colore "illuminato" o "ombra" ed altri; il parametro passato in
SA_Pens
è un array di WORD che contiene in una determinata posizione, che identifica di quale colore si parla, il codice della
penna
corrispondente;
l'array deve terminare con ~0, per cui se si vuole utilizzare la definizione standard delle
penne
, basta specificare un array con solo ~0;
se invece occorre utilizzare particolari colori, bisogna specificarli nell'array seguendo la tabella delle definizioni (vedere riquadro). L'array delle
penne
di uno schermo (dri_Pens) è contenuto nella struttura DrawInfo, che contiene tutte le informazioni per il corretto rendering dello schermo e del suo contenuto (dati sui font ecc.).
Per ottenere il puntatore alla struttura DrawInfo di uno schermo, bisogna utilizzare la funzione GetScreenDrawInfo con parametro puntatore alla struttura
Screen
; una volta utilizzata, bisogna liberare tale struttura con FreeScreenDrawInfo.

La visuale

Nell'apertura dello schermo si può specificare un diverso tipo di overscan tra i seguenti:
OSCAN_TEXT
OSCAN_STANDARD
OSCAN_MAX
OSCAN_VIDEO
Occorre quindi conoscere qual è la reale risoluzione dello schermo; per questo si può utilizzare la funzione QueryOverscan:

```
ris = QueryOverscan(screen_modeID,rettangolo,tipooverscan);
```

dove "ris" è una LONG che indica successo se ha un valore diverso da zero; "screen_modeID" è la ULONG che identifica il tipo di schermo; "rettangolo" è il puntatore ad una struttura Rectangle contenente i limiti dello schermo; attenzione, la struttura Rectangle non verrà creata dalla

funzione, ma dovrete crearla voi e passare alla funzione il suo indirizzo (mediante l'operatore &) in modo che la funzione possa modificarne i campi; il parametro "tipooverscan" specifica il tipo di overscan, scelto fra uno tra quelli descritti prima.

1.23 Attributi dello schermo

Attributi dello schermo

Qui di seguito vengono riportati gli attributi impostabili dall'applicazione all'apertura dello schermo, mediante i tag qui elencati:

SA_ErrorCode
Codice d'
errore
; il campo Data del tag è il puntatore ad una long in cui verrà riportato il codice dell'errore che si è verificato nel caso in cui non sia stato possibile aprire lo schermo (OpenScreenTags e OpenScreenTagList ritornano in tal caso NULL).

SA_Left, SA_Top
Posizione inziale dello schermo rispetto al rettangolo di text overscan.

SA_Width, SA_Height
Grandezza dello schermo; si possono passare valori numerici non negativi o delle costanti predefinite STDSCREENWIDTH o STDSCREENHEIGHT che corrispondono a valori di grandezza pari al display clip attuale

SA_Depth
Profondità in bitplanes dello schermo; default è un bitplane, il numero massimo può dipende dalla modalità grafica scelta.

SA_DisplayID
Chiave di identificazione della modalità di visualizzazione (esempio NTSC, PAL, DBLPAL ecc.); lo standard è PAL e tale chiave, insieme a tutte le informazioni riguardanti la modalità sono dipsonibile nel display database della graphics.library (lo vedremo quando tratteremo la graphics.library)

SA_Pens
Specificazione delle penne utilizzate per disegnare schermo, finestre e gadgets; il campo data è un puntatore ad un array di UWORD terminante con ~0; questa tecnica permette di ottenere il look 3D del sistema 2.0

SA_DetailPen
E' il colore di penna del primo piano per il disegno della barra titolo e menù

SA_BlockPen
E' il colore di penna dello sfondo per il disegno della barra titolo e menù

SA_Title
Titolo di default dello schermo; il campo data è un puntatore a stringa; questo è il titolo visualizzato quando la finestra selezionata non ha titolo o quando nessuna finestra sullo schermo è selezionata

SA_Colors
Specifica la palette di colori dello schermo; il campo data è il puntatore ad un array di strutture ColorSpec terminante con una struttura ColorSpec con ColorIndex uguale a -1

SA_FullPalette
Inizializza la palette di colori con quella definita nel preferences; il capo data è di tipo booleano, TRUE per settare la palette preferences; per default è FALSE

SA_Font
Il campo data è il puntatore ad una struttura TextAttr (definita in <graphics/text.h>) che specifica tipo, grandezza e stile del font dello schermo

SA_SysFont
ALternativa a SA_Font; permette di selezionare uno dei font del preferences di sistema; Data è un LONG senza segno e può avere i seguenti valori:
0 - apre lo schermo con il default font a grandezza fissa
1 - apre lo schermo con il font scelto dall'utente che può essere proporzionale
E' illegale cambiare il font dello schermo dopo la sua apertura; il titolo, i menù ed i testi dei gadgets utilizzano il font dello schermo.

SA_Type
Indicano il tipo di schermo e può vale CUSTOMSCREEN o PUBLICSCREEN (WBENCHSCREEN è riservato per usi del sistema)

SA_BitMap
Utilizza una bitmap specificata e creata dall'applicazione, per lo schermo

SA_Behind
E' un valore booleano che se TRUE (di default è FALSE) apre lo schermo dietro a tutti gli altri

SA_Quiet
Se TRUE (default FALSE) diabilita il rinfresco dello schermo da parte di Intuition

SA_ShowTitle
Se TRUE pone il titolo dello schermo davanti ad ogni backdrop window

SA_AutoScroll
Se TRUE verrà abilitato l'autoscroll per questo schermo; autoscroll significa che, se lo schermo è più grande della zona visibile ed il puntatore arriva al limite della zona visibile e viene spinto più in là (anche se il puntatore rimane fermo) lo schermo scrolla automaticamente per mostrare la zona nascosta

SA_PubName
La presenza di questo tag significa che lo schermo è di tipo pubblico (SA_Type deve essere impostato a PUBLICSCREEN); il campo data del tag è il puntatore ad una stringa che è il nome dello schermo pubblico che può essere utilizzato da altre applicazioni.

SA_PubSig, SA_PubTask
Task ID (ritornato da FindTask) e segnale per comunicare al task che

```
l'ultima finestra sullo schermo pubblico è stata chiusa e che, quindi lo
schermo può essere chiuso; data per SA_PubSig è un LONG, per SA_PubTask è il
puntatore alla struttura Task identificante il Task che dovrà essere
segnalato; questi due tags devono essere specificati dopo SA_PubName

SA_Overscan
Specifica una dei OSCAN_tipo, standard overscan del sistema; default è
OSCAN_TEXT; non specificare questo tag se viene utilizzato SA_DClip

SA_DCLip
Definizione della regione visibile; il campo data è un puntatore ad una
struttura Rectangle che definisce la regione visibile dello schermo
```

1.24 Codici d'errore dello schermo

```
Codici d'errore ritornati utilizzando l'attributo SA_ErrorCode:

OSERR_NOMONITOR     - monitor per modalità di schermo non disponibile.
OSERR_NOCHIPS       - versione di chip-custom troppo vecchia per modalità di ↩
   schermo.
OSERR_NOMEM         - non si può allocare abbastanza memoria
OSERR_NOCHIPMEM     - non si può allocare abbastanza memoria 'chip'
OSERR_PUBNOTUNIQUE  - nome di schermo pubblico già usato
OSERR_UNKNOWNMODE   - modalità richiesta non conosciuta
OSERR_TOODEEP       - schermo troppo profondo per essere aperto con questo ↩
   hardware (V39)
OSERR_ATTACHFAIL    - è stata richiesta una connessione illegale di schermi (V39)
```

1.25 La struttura Screen

```
La struttura Screen

Qui di seguito viene riportata la struttura Screen con la spiegazione dei
campi più importanti:

struct Screen
{
  struct Screen *NextScreen;
  struct Window *FirstWindow;
  WORD LeftEdge,TopEdge,Width,Height;
  WORD MouseY,MouseX;
  UWORD Flags;
  UBYTE *Title,*DefaultTitle;
  BYTE BarHeight,BarVBorde,BarHBorder,MenuVBorder,MenuHBorder;
  BYTE WBorTop,WBorLeft,WBorRight,WBorBottom;
  struct TextAttr *Font;
  struct ViewPort ViewPort;
  struct RastPort RastPort;
  struct BitMap BitMap;
  struct Layer_Info LayerInfo;
  struct Gadget *FirstGadget;
  UBYTE DetailPen,BlockPen;
```

```
  UWORD SaveColor0;
  struct Layer *BarLayer;
  UBYTE *ExtData,*UserData;
}
```

LeftEdge, TopEdge
le variabili LeftEdge e TopEdge indicano la posizione dello schermo relativa all'angolo in alto a sinistra del display visibile del monitor; valori positivi indicano posizioni verso destra e verso il basso e negative il contrario; pria della versione V36 del s.o. i valori di LeftEdge venivano ignorati e valori negativi del TopEdge venivano considerati illegali

MouseX, MouseY
le coordinate del puntatore del mouse relative all'angolo in alto a sinistra dello schermo

ViewPort, RastPort, BitMap e LayerInfo
strutture della graphics.library per la gestione grafica e il rinfresco dello schermo

BarLayer
il puntatore alla struttura Layer per la barra titolo

WBorTop, WBorLeft, WBorRight, WBorBottom
valori dei bordi della finestra nello schermo

Font
il font di default dello schermo

UserData
di utilizzo libero per l'applicazione

1.26 Le finestre (window)

Le finetsre (window)

La finestra è una zona rettangolare dello schermo che funziona come un piccolo terminale, e permette di interagire con l'utente; la window dispone di diversi gadgets (bottoni) standard di sistema che possono essere specificati dal programmatore all'apertura di quest'ultima:

Close Gadget:
Il gadget in alto a sinistra che permette la chiusura della finestra

Drag Bar:
La barra titolo che permette il trascinamento della finestra

Zoom Gadget:
Il gadget a sinistra tra quelli in alto a destra, permette di scegliere tra due dimensioni della finestra (se la finestra è abilitata a cambiare la sua grandezza)

Depth Gadget:
Il gadget a destra fra quelli in alto a destra, permette di far passare avanti o dietro la finestra, rispetto a tutte le altre

```
Sizing Gadget:
Il gadget in basso a destra, permette di cambiare la dimensione della
finestra.

La finestra può essere attiva (active), nel qual caso il bordo viene colorato
di blu; esiste una sola finestra attiva per volta, dato che selezionando
una finestra l'utente decide l'input-focus (vale a dire su quale finestra
far convergere i dati di ingresso quali tastiera, mouse ecc.).
Per aprire la finestra esistono tre diverse funzioni di intuition, che
ricoprono gli stessi ruoli di quelle viste per lo schermo:

Finestra = OpenWindowTagList(NuovaFinestra,listatags);
Finestra = OpenWindowTags(NuovaFinestra,tagId1,tagVal1,...);
Finestra = OpenWindow(NuovaFinestra);

"Finestra" è il puntatore alla struttura
                Window
                 relativa alla finestra aperta
(viene ritornato NULL in caso non sia stato possibile aprirla);
"NuovaFinestra" è il puntatore alla struttura NewWindow o ExtNewWindow
contenente i parametri della finestra da aprire (dal 2.0 non ha più
importanza utilizzare questo parametro); mentre "listatags" in
OpenWindowTagList o "tagId1,tagVal1,..." in OpenWindowTags sono liste di tag
per i
                parametri
                 della finestra passate nelle due maniere viste la volta
scorsa; OpenWindow è la vecchia funzione per l'apertura di una finestra che,
utilizzando la struttura ExtNewWindow con flag
                WFLG_NW_EXTENDED
                 attivato
permette la compatibilità con versioni del sistema inferiore al v36.
Per chiudere una finestra aperta, bisogna
chiamare la funzione:

CloseWindow(Finestra);

dove "Finestra" è il puntatore alla struttura
                Window
                 ritornata da
OpenWindowTags. Per indicare su quale schermo aprire la finestra, occorre
specificare nella OpenWindowTags il tag
                WA_CustomScreen
                 con il puntatore allo
schermo come parametro; se invece si vuole aprire la finestra su schermo
pubblico si può indicare il tag WA_PubScreen con il puntatore allo schermo
pubblico (ottenuto con LockPubScreen), oppure utilizzando il tag

                WA_PubScreenName
                 e passando il nome dello schermo pubblico; se non viene
specificato nessun tag che indichi quale schermo utilizzare, oppure

                WA_PubScreenName
                 con parametro NULL, la finestra verrà aperta sullo schermo
pubblico di default (normalmente il WorkBench).
Esistono tre tipi particolari di finestre, che possono essere impostati
mediante gli attributi all'apertura di quest'ultime; da notare che un tipo
```

non esclude l'altro, per cui una finestra può appartenere a tutte e tre le categorie e beneficiare di tutte le caratteristiche.

BackDrop window
Vale a dire finestra che "cade" dietro; una finestra con questa caratteristica risulta sempre dietro alle altre, anche quando l'utente agisce sul gadget di profondità; la finestra può però essere davanti ad altre di tipo backdrop. L'unico gadget di sistema che questo tipo di finestre può avere è quello di chiusura, mentre non vi sono limitazioni per gadgets creati dall'applicazione. Questo tipo di finestre può servire ad inserire delle immagini di sottofondo che sembrino disegnate direttamente sullo schermo (soprattutto se la finestra è anche di tipo borderless); per attivare questa caratteristica utilizzare l'attributo

```
WA_Backdrop
.
```

BorderLess window
Una finestra borderless è una finestra senza bordi; vale a dire che i bordi non esistono e non vengono rinfrescati, per cui evitate di utilizzare il titolo per la finestra, o di utilizzare gadget di sistema, poiché ciò provocherebbe il rinfresco di alcune zone dei bordi; evitate di utilizzare finestre borderless che coprano tutto lo schermo, perché potrebbero confondere l'utente; un applicazione tipica è quella accennata prima con le finestre backdrop; per utilizzare questa caratteristica impostare l'attributo

```
WA_Borderless
.
```

GimmeZeroZero window
Le finestre GimmeZeroZero provvedono a creare un layer (la struttura per la memorizzazione dell'immagine e per il suo rinfresco) separato per il bordo; normalmente se il programmatore disegna o compie una qualunque operazione grafica sulla finestra, deve tener conto del bordo (infatti scrivendo il pixel di cordinate 0,0 si interviene nel primo pixel in alto a sinistra sul bordo); invece se la finestra è di tipo GimmeZeroZero, il programmatore può intervenire solo sul contenuto della finestra senza intaccare il bordo; attivando il GimmeZeroZero in una finestra si aumentano i tempi di rinfresco video, in quanto il sistema deve gestire due layers per una finestra; per attivare questa caratteristica, utilizzare il tag

```
WA_GimmeZeroZero
.
```

1.27 Attributi della finestra

```
Attributi della finestra
```

Elenchiamo e descriviamo tutti i tag utilizzabili da OpenWindowTags:

WA_Left,WA_Top,WA_Width,WA_Height
Indicano la posizione e la grandezza della finestra; questi valori sono relativi all'angolo in alto a sinistra dello schermo a cui corrisponde il valore 0,0; queste variabili equivalgono ai campi della struttura NewWindow: Left,Top,Width,Height, utilizzata nella funzione OpenWindow

WA_DetailPen,WA_BlockPen
Sono i colori di primo piano e sfondo per il disegno della barra titolo, del testo, dei gadgets ecc.; questi valori sono quasi inutilizzati con l'introduzione del DrawInfo dalla versione V36 del sistema

WA_IDCMP
Indica per quali eventi di input, che accadono mentre la finestra è attiva, l'applicazione deve essere avvisata

WA_Gadgets
Il puntatore alla prima struttura Gadget della lista dei gadgets della finestra

WA_CheckMark
Il puntatore alla struttura Image dell'immagine da utilizzare come checkmark nei menù

WA_Title
Puntatore alla stringa di caratteri del titolo della finestra

WA_ScreenTitle
Puntatore alla stringa di caratteri del titolo da visualizzare sulla barra dello schermo, quando la finestra è attiva

WA_CustomScreen
Puntatore allo schermo custom su cui aprire la finestra

WA_MinWidth,WA_MinHeight,WA_MaxWidth,WA_MaxHeight
Questi valori indicano la grandezza minima e massima, che la finestra può assumere se l'utente può cambiare grandezza (è impostato il flag WFLG_WINDOWSIZING); se la finestra non possiede il gadget per il cambiamento di grandezza, allora questi valori sono ignorati. Se per i valori di grandezza massima, viene utilizzato ~0 allora la finestra può assumere la maggior grandezza possibile; se il valore utilizzato per uno di questi campi è 0, allora viene considerato come valore quello impostato inizialmente (da WA_Width o WA_Height)

WA_InnerWidth,WA_InnerHeight
Specificano la grandezza della zona interna della finestre, indipendentemente dalla grandezza dei bordi; può essere utile, specificando questi valori, impostare il flag WA_AutoAdjust

WA_PubScreen
Puntatore allo schermo pubblico su cui aprire la finestra

WA_PubScreenName
Puntatore alla stringa indicante il nome dello schermo pubblico, su cui aprire la finestra

WA_PubScreenFallBack
Un valore booleano che, se impostato a TRUE indica di aprire la finestra sullo schermo pubblico di default se non è disponibile, quello specificato in WA_PubScreenName

WA_Zoom
Un puntatore ad un array di 4 WORD che indicano i valori iniziali di LeftEdge, TopEdge, Width, Height per la posizione e grandezza della finestra

da utilizzare, quando viene attivato il gadget di "zoom"

WA_MouseQueue
Il numero massimo di eventi da mouse da mantenere in coda di input; questo valore può essere modificato da SetMouseQueue(), una volta aperta la finestra

WA_RptQueue
Il numero massimo di eventi da tastiere da mantenere in coda di input

Attributi booleani:

WA_SizeGadget
Se TRUE, indica ad intuition di aprire la finestra con il gadget di cambiamento dimensione, in basso a destra

WA_SizeBRight
Pone il gadget di grandezza nel bordo di destra

WA_SizeBBottom
Pone il gadget di grandezza nel bordo di sotto

WA_DragBar
Trasforma la barra titolo della finestra, in un gadget che permette di trascinarla

WA_DepthGadget
Indica di utilizzare il gadget di profondità, per portare la finestra avanti o indietro rispetto a tutte le altre

WA_CloseGadget
Indica di inserire il gadget di chiusura della finestra

WA_ReportMouse
Spedisce eventi di spostamento del mouse, come coordinate (x,y). Questo flag può essere modificato direttamente, una volta che la finestra è aperta, utilizzando queste due chiamate:
window->flags |= WFLG_REPORTMOUSE; /* attiva il flag */
window->flags &= ~WFLG_REPORTMOUSE; /* disattiva */
dove "window" è il puntatore alla finestra

WA_NoCareRefresh
Se TRUE, impedisce ad intuition di inviare l'evento IDCMP_REFRESHWINDOW, quando necessita il rinfresco della finestra

WA_Borderless
Indica di aprire una finestra senza bordi

WA_Backdrop
Questa finestra deve essere backdrop

WA_GimmeZeroZero
Quest finestra è di tipo GimmeZeroZero; finestre GimmeZeroZero hanno il bordo conservato e rinfrescato in un Layer (struttura per il mantenimento di blocchi grafici) a parte; ciò rallenta le operazioni di rinfresco

WA_Activate
Attiva la finestra appena aperta

WA_RMBTrap
Intercetta l'evento di selezione del tasto destro del mouse; con questo flag, i menù vengono disabilitati e viene inviato un messaggio di tipo IDCMP_MOUSEBUTTONS (se specificato nel tag WA_IDCMP) con il codice del tasto destro

WA_SimpleRefresh
L'applicazione ha il compito per il completo rinfresco del contenuto dell'immagine (vedere Rinfreshi della finestra)

WA_SmartRefresh
L'applicazione deve rinfrescare solo le zone scoperte da un allargamento della finestra, il resto viene mantenuto e rinfrescato da sistema

WA_SuperBitMap
Questo è il puntatore ad una struttura BitMap contenente l'immagine del contenuto della finestra; in tal caso il sistema rinfrescherà anche le zone eventualmente scoperte da un cambiamento di grandezza della finestra dato che utilizza direttamente il BitMap specificato

WA_AutoAdjust
Permette ad intuition di "aggiustare" la posizione e la grandezza della finestra, per farla cacciare nello schermo; viene prima modificata la posizione e poi la grandezza; ha particolarmente senso se vengono specificati WA_InnerWidth o WA_InnerHeight

WA_MenuHelp (dalla V37 del s.o.)
Abilita IDCMP_MENUHELP; premendo il tasto HELP mentre si selezione un menù, viene inviato il messaggio appena indicato, in modo che l'applicazione possa fornire un aiuto sull'utilizzo del menù

WA_Flags
Per l'inizializzazione multipla di diversi
flag
appena descritti, mediante
le definizioni WFLG_flag OR-ate fra loro; questo è un tag obsoleto (vale a dire che il suo uso può e conviene che venga sostituito con i tag appena visti)

WA_BackFill
Permette di specificare una funzione di backfill hook per il layer della finestra; questa è una funzione di rinfresco del layer creata dal programmatore in casi particolari di rinfresco

1.28 Penne del Drawinfo

Penne del Drawinfo

Vengono elencate le costanti indicanti la funzione di una determinata penna; la costante indica una posizione del vettore Pens, per cui il valore della penna che indica DETAILPEN ad esempio deve essere ricavato in questo modo: DrawInfo->dri_Pens[DETAILPEN]

DETAILPEN

```
Usata per la compatibilità con i vecchi sistemi; utilizzata per disegnare il
testo e la barra titolo dello schermo

BLOCKPEN
Usata per la compatibilità con i vecchi sistemi; utilizzata per la barra
titolo dello schermo

TEXTPEN
Penna per il colore del testo regolare, visualizzato sul BACKGROUNDPEN

SHINEPEN
Penna per il colore "illuminato" del bordo 3D

SHADOWPEN
Penna per il colore "ombra" del bordo 3D

FILLPEN
Penna per colorare la finestra attiva e gadget selezionati

FILLTEXTPEN
Penna per disegnare il testo su FILLPEN

BACKGROUNDPEN
Penna per il colore di sfondo; attualmente deve essere 0

HIGHLIGHTTEXTPEN
Penna per colore "speciale" o testo super illuminato su BACKGROUNDPEN
```

1.29 La struttura Window

```
                La struttura Window


Qui di seguito viene descritta la struttura Window con tutti i suoi campi più
importanti:

struct Window
{
    struct Window *NextWindow;          /* puntatore alla prossima finestra */
    WORD LeftEdge,TopEdge,Width,Height;
    WORD MouseY,MouseX;
    WORD MinWidth, MinHeight;           /* grandezza minima */
    UWORD MaxWidth, MaxHeight;          /* grandezza massima */
    ULONG
                Flags
                ;                        /* flags della finestra */
    struct Menu *MenuStrip;             /* puntatore al menù */
    UBYTE *Title;                       /* titolo di questa finestra */
    struct Requester *FirstRequest;     /* puntatore ai Requesters attivi */
    struct Requester *DMRequest;        /* double-click Requester */
    WORD ReqCount;
    struct Screen *WScreen;
    struct RastPort *RPort;
    BYTE BorderLeft,BorderTop,BorderRight,BorderBottom;
    struct RastPort *BorderRPort;
```

```
    struct Gadget *FirstGadget;         /* gadgets della finestra */
    struct Window *Parent, *Descendant;
    /* informazioni sullo sprite del puntatore della finestra */
    UWORD *Pointer;                     /* dati dello sprite */
    BYTE PtrHeight;                     /* altezza dello sprite */
    BYTE PtrWidth;                      /* larghezza dello sprite */
    BYTE XOffset, YOffset;              /* offsets dello sprite */
    struct MsgPort *UserPort,*WindowPort; /* porte della finestra */
    struct IntuiMessage *MessageKey;
    UBYTE DetailPen,BlockPen;   /* per la colorazione di bar/border/gadget */
    struct Image *CheckMark;            /* immagine checkmark per i menù */
    UBYTE *ScreenTitle; /* titolo dello schermo quando la finestra è attiva */
    WORD GZZMouseX,GZZMouseY,GZZWidth,GZZHeight;
    UBYTE *ExtData;
    BYTE *UserData;
    struct Layer *WLayer;
    struct TextFont *IFont;
    ULONG MoreFlags;                    /* altri flags (dalla V36) */
};
```

LeftEdge,TopEdge,Width,Height
Queste variabili contengono le attuali informazioni di posizione e dimensione della finestra

MouseX,MouseY,GZZMouseX,GZZMouseY
Indicano la posizione del puntatore del mouse, rispetto all'angolo in alto a sinistra della finestra; per le finestre GimmeZeroZero le variabili GZZ indicano la posizione relativa al layer interno (vedere tipi di finestra), altrimenti indica la posizione relativa all'angolo in alto a sinistra della finestra dopo il bordo

ReqCount
Contiene il numero dei rquesters correntemente visualizzati nella finestra

WScreen
Il puntatore allo schermo su cui la finestra è visualizzata

RPort
Puntatore alla struttura RastPort della finestra (la struttura RastPort serve per disegnare e in genere per tutte le operazioni grafiche, vedremo meglio questo argomento in una delle prossime puntate)

BorderLeft,BorderTop,BorderRight,BorderBottom
Indicano l'attuale grandezza dei bordi della finestra (da sinistra, sopra, destra e sotto)

BorderRPort
Con le finestre GimmeZeroZero, indica il puntatore alla RastPort del layer esterno

UserData
Puntatore di utilizzo libero per l'applicazione

1.30 Flag della finestra

Flag della finestra

Riportiamo in questa tabella tutti i possibili flag utilizzabili nel tag WA_Flags all'apertura della finestra; ricordiamo che per indicare più flags bisogna effettuare l'operazione di or bit a bit; esempio: WFLG_DRAGBAR|WFLG_SMART_REFRESH.

WFLG_SIZEGADGET
La finestra possiede il gadget di ridimensionamento

WFLG_DRAGBAR
La finestra possiede la barra di spostamento

WFLG_DEPTHGADGET
La finestra possiede il gadget di profondità

WFLG_CLOSEGADGET
La finestra possiede il gadget di chiusura

WFLG_SIZEBRIGHT
La finestra usa il bordo destro

WFLG_SIZEBBOTTOM
La finestra usa il bordo inferiore

WFLG_SMART_REFRESH
La finestra usa il rinfresco SMART

WFLG_SIMPLE_REFRESH
La finestra usa il rinfresco SIMPLE

WFLG_SUPER_BITMAP
La finestra è di tipo SuperBitMap

WFLG_OTHER_REFRESH
La finestra usa altri rinfreschi (per usi futuri nel caso di realizzazione di nuovi tipi di rinfresco)

WFLG_BACKDROP
La finestra è di tipo backdrop

WFLG_REPORTMOUSE
Specifica di avvisare per ogni movimento del mouse

WFLG_GIMMEZEROZERO
La finestra è di tipo GimmeZeroZero

WFLG_BORDERLESS
La finestra è di tipo borderless

WFLG_ACTIVATE
La finestra si attiva automaticamente all'apertura

WFLG_RMBTRAP
Intercetta l'evento di pulsante destro del mouse in modo da farlo gestire all' applicazione senza visualizzare automaticamente la barra dei menù

```
WFLG_NOCAREREFRESH
Non avvisare per i rinfreschi della finestra

WFLG_NW_EXTENDED (V36)
La struttura per l'apertura della finestra è ExtNewWindow per la compatibilità con ↩
    il vecchio sistema

WFLG_NEWLOOKMENUS (V39)
La finestra possiede menù con nuovo look
```

1.31 Comunicazione con Intuition: IDCMP

```
                Comunicazione con Intuition: IDCMP

Come abbiamo detto la puntata scorsa, la finestra rappresenta il luogo in
cui il programma visualizza i dati per l'utente, ed il luogo in cui l'utente
indica al programma quale operazione deve compiere; mentre le prime azioni
sono svolte dalla libreria grafica e da alcune funzioni di Intuition che
permettono di visualizzare testi e grafici sulla finestra, le operazioni di
input su quest'utlima sono realizzate da IDCMP.
IDCMP è un acronimo che significa Intuition Direct Communications Message Port,
vale a dire porta messaggi per la comunicazione diretta con intuition; in altre
parole intuition permette di avvissare e comunicare qualsiasi input (mouse,
finestra, gadget, menù ecc.) avvenuto in una determinata finestra,
all'applicazione; come avete potuto capire dal nome, IDCMP è una porta
messaggi exec associata ad una finestra, che riceve un formato particolare
di messaggio,
                IntuiMessage
                 contenente oltre alle informazioni del messaggio
exec, il tipo di evento di input, altri eventuali dati su di esso, su quale
finestra è avvenuto ed altro ancora. Il programmatore deve specificare,
all'atto dell'apertura della finestra, con il tag
                WA_IDCMP
                 per quali
messaggi deve essere avvisato mediante IDCMP; in tale maniera posso
chiedere di essere semplicemente ratificato di eventi sul menù, perché devo
attuare operazioni solo in risposta di questo tipo di evento; se tale campo
è diverso da NULL la porta IDCMP che riceve questi
                messagg
                i di intuition,
viene creata automaticamente al momento dell'apertura della finestra; se
tale tag è NULL non viene aperta la porta IDCMP. Questo vuol dire che ogni
finestra può avere una propria porta, ed ognuna di queste riceve i messaggi
indicati all'atto dell'apertura, quando questa è attiva. E' comunque
possibile modificare gli eventi per i quali essere avvisati, mediante la
seguente funzione:

Successo = ModifyIDCMP(Finestra,IDCMPFlags);

Dove "Finestra" è il puntatore alla struttura Window della finestra;

                IDCMPFlags
                 è una ULONG dello stesso tipo passato in
                WA_IDCMP
```

```
                 indicante
quali eventi di input gestire; Successo (utilizzato a partire da V37) se
vale NULL vuol dire che non è stato possibile modificare l'IDCMP; dato che,
se la precedente configurazione poteva essere NULL e modificando
l'
                IDCMPFlags
                 impostando almeno un evento il sistema provvede direttamente
da ModifyIDCMP a creare la porta IDCMP, può accadere che non venga aperta
(mancanza di memoria o altro); il puntatore alla porta creata dal sistema è
presente nel campo UserPort della struttura Window.
Tirando le somme, il nucleo di ogni applicazione si occupa di attendere dalla
porta IDCMP messaggi di Intuition (utilizzando Wait o WaitPort come visto in
una delle precedenti puntate) e di rispondere con determinate azioni quando
questi arrivano; ad esempio se si seleziona l'opzione "Open" del menù,
bisogna far apparire il file requester per scegliere il file da aprire; si
può realizzare (vedere listati) una routine che si occupa proprio di
attendere un messaggio.
Come abbiamo prima accennato ogni finestra aperta può avere
una propria porta IDCMP; questo non è necessario, poiché se un'applicazione
apre più di una finestra, può gestire tutti i messaggi con un'unica porta,
dato che nella struttura
                IntuiMessage
                 vi è anche il campo per determinare
da quale finestra arriva il messaggio; in altre parole la porta IDCMP viene
condivisa fra più finestre; per fare questo bisogna effettuare i seguenti
passi:
1 - Creare una porta messaggi exec (che sarà utilizzata come porta IDCMP)
    mediante CreatePort o CreateMsgPort
2 - Aprire la finestra con il tag
                WA_IDCMP
                 impostato a NULL; questo impedirà
    ad Intuition di aprire una porta per quella finestra
3 - Inserire il puntatore della porta creata nel punto 1 nel campo UserPort
    della finestra appena aperta
4 - Chiamare ModifyIDCMP() per impostare i corretti
                IDCMPFlags
                 per quella
    finestra; attenzione a non chiamare ModifyIDCMP(finestra,NULL) poiché si
    chiuderebbe la porta condivisa
5 - Quando l'applicazione decide di chiudere una finestra con una porta IDCMP
    condivisa, bisogna che vengano prima risposti tutti i messaggi, presenti
    nella porta, di quella finestra; esiste per questo una funzione
    denominata CloseWindowSafely, riportata sul ROM Kernel Manual che elimina
    dalla porta tutti i messaggi prima di chiudere la finestra (vedere
    listati); la funzione dopo aver eliminato tutti i messaggi dalla porta,
    assegna al campo UserPort della finestra NULL e chiama
    ModifyIDCMP(finestra,NULL) in modo da interrompere il flusso di messaggi
    senza liberare la porta (che potrebbe servire ad altre finestre)
6 - Una volta chiuse tutte le finestre, bisogna deallocare la porta mediante
    DeletePort (se è stata usata CreatePort) o DeleteMsgPort (se è stata
    usata CreateMsgPort).
```

1.32 IDCMP FLags

IDCMP Flags

Riportiamo i flag indicanti i diversi eventi di input; quasti flag vengono usati nel campo IDCMPFlags della finestra,~nel tag WA_IDCMP nella funzione OpenWindowTagList e come identificatori del tipo di evento di un messaggio ricevuto; per specificare più eventi utilizzare l'operatore logico OR (|);
esempio:
ModifyIDCMP(finestra,IDCMP_MOUSEBUTTONS|IDCMP_GADGETUP);

IDCMP_REQVERIFY
avvisa l'applicazione che un requester sta per aprirsi nella finestra, ed il sistema attende che l'applicazione risponda prima di aprirlo

IDCMP_REQCLEAR
avvisa l'applicazione quando un requester viene rimosso dalla finestra, in modo che il programma possa rinfrescare correttamente il suo contenuto

IDCMP_REQSET
avvisa l'applicazione che un requester è stato appena aperto nella finestra; questo evento è diverso da IDCMP_REQVERIFY in quanto avvisa quando il requester è stato oramai aperto; attenzione a rinfrescare o disegnare nella finestra solo quando tutti i requester sono stati chiusi (vale a dire per ogni IDCMP_REQSET è stato spedito un IDCMP_REQCLEAR)

IDCMP_MENUVERIFY
questo evento avvisa l'applicazione che il sistema sta per attivare la barra menù o i menù; può necessitare essere avvisati se si stanno eseguendo dei disegni "critici" direttamente su schermo; vale a dire devono essere ultimati prima di qualunque altro intervento; il sistema attende che l'applicazione risponda, prima di attivare i menù

IDCMP_SIZEVERIFY
avvisa l'applicazione che l'utente sta per cambiare la dimensione della finestra (ovviamente la finestra deve avere il gadget di cambiamento dimensione), permettendo di ultimare operazioni "critiche"; il sistema attenderà la risposta del messaggio prima di procedere al cambiamento di grandezza

Attenzione per tutti gli eventi "VERIFY" (IDCMP_MENUVERIFY, IDCMP_SIZEVERIFY ecc.): rispondere il prima possibile al messaggio in quanto l'utente viene bloccato ed attende (e il multitasking va a farsi una passeggiata); non utilizzare MAI funzioni AmigaDOS mentre è attivo un evento "VERIFY"; dalla V36 del sistema se l'applicazione non risponde per un certo periodo di tempo specificato dall'utente, l'evento "VERIFY" viene interrotto. Conviene utilizzare questi eventi con parsimonia, eliminando l'avviso quando non serve, con ModifyIDCMP

IDCMP_NEWSIZE
avvisa l'applicazione quando la finestra ha subito un cambio di dimensione; vedere anche IDCMP_CHANGEWINDOW

IDCMP_REFRESHWINDOW
avvisa l'applicazione che la finestra ha bisogno di essere rinfrescata (quando passa da dietro un gruppo di finestre in avanti o quando viene ingrandita); necessita quando il rinfresco del contenuto della finestra è gestito dall'applicazione. Questo evento ha senso solo con finestre con refresh WFLG_SIMPLE_REFRESH e WFLG_SMART_REFRESH

IDCMP_MOUSEBUTTONS
avvisa l'applicazione che uno dei pulsanti del mouse è stato premuto (a meno che, questo non significhi qualcosa di particolare come la pressione di un gadget)

IDCMP_MOUSEMOVE
funziona solamente se è stato impostato il flag WFLG_REPORTMOUSE della finestra o il flag GACT_FOLLOWMOUSE in uno dei gadgets. Allora ogni volta che il mouse si mouve, verrà inviato questo messaggio.

IDCMP_GADGETDOWN
avvisa che l'utente ha selezionato un gadget, che tu hai creato con il flag GACT_IMMEDIATE.

IDCMP_GADGETUP
avvisa che l'utente ha rilasciato un gadget, che tu hai creato con il flag GACT_RELVERIFY

IDCMP_MENUPICK
l'utente ha probabilmente selezionato un'opzione del menù

IDCMP_CLOSEWINDOW
avvisa che l'utente ha selezionato il gadget di sistema di chiusura della finestra

IDCMP_RAWKEY
avvisa che l'utente ha premuto un tasto; questo evento riporta il codice assoluto (RAW) del tasto; il codice del tasto è passato nel campo Code dell'IntuiMessage

IDCMP_VANILLAKEY
avvisa che l'utente ha premuto un tasto e ritorna il codice di quest'ultimo; però il codice viene prima trasformato sequendo la keymap in formato ANSI. Tasti come HELP non possono essere gestiti con IDCMP_VANILLAKEY (usare IDCMP_RAWKEY). Dalla V36 del sistema se hai impostato sia IDCMP_RAWKEY che IDCMP_VANILLAKEY il sistema avviserà con un evento IDCMP_RAWKEY solo per quei tasti speciali; per tutti gli altri tasti invierà IDCMP_VANILLAKEY unendo così i vantaggi dell'uno e dell'altro

IDCMP_INTUITICKS
viene inviato questo messaggio approssimativamente 10 volte al secondo; può servire per temporizzazione per alcuni operazioni (come il double-click)

IDCMP_DELTAMOVE
fornisce i valori delta X/Y di movimento del mouse; può servire a controllare l'accelerazione del mouse; se vengono impostati contemporaneamente IDCMP_MOUSEMOVE e IDCMP_DELTAMOVE, si otterranno messaggi IDCMP_MOUSEMOVE con i valori delta x e y nei campi MouseX e MouseY dell'IntuiMessage

IDCMP_NEWPREFS
avvisa che sono state cambiate le preferenze di sistema

IDCMP_ACTIVEWINDOW e IDCMP_INACTIVEWINDOW
avvisano che la finestra viene attivata o disattivata rispettivamente

```
IDCMP_DISKINSERTED e IDCMP_DISKREMOVED
avvisano che un disco è stato inserito o rimosso rispettivamente

IDCMP_IDCMPUPDATE
nuovo dalla V36, usato come un canale di comunicazione da gadget custom e
boopsi alla tua pplicazione

IDCMP_CHANGEWINDOW
nuovo dalla V36, avvisa che le dimensioni o la posizione della finestra
vengono cambiate dall'utente o da una funzione intuition

IDCMP_MENUHELP
nuovo dalla V37; se specifichi WA_MenuHelp all'apertura della finestra,
quando l'utente preme il tasto HELP mentre seleziona un menù, viene inviato
questo evento in modo che l'applicazione possa fornire un aiuto all'utente

IDCMP_GADGETHELP
nuovo dalla V39 del sistema. Se attivi il gadget help della tua finestra
usando HelpControl(), Intuition spedirà IDCMP_GADGETHELP quando il mouse
passa sopra determinati gadget della tua finestra
```

1.33 IntuiMessage

```
IntuiMessage

Viene qui descritta la struttura messaggi per la porta IDCMP; dalla versione
V39 del sistema è presenta una nuova struttura denominata ExtIntuiMessage
con ulteriori dati per nuovi usi

struct IntuiMessage
{
  struct Message ExecMessage;  /* header per il messaggio exec */
  ULONG Class;  /* indica il tipo di evento che ha generato il messaggio;
                   sono gli stessi flag utilizzati in WA_IDCMP */
  UWORD Code;  /* indica un codice o dato del messaggio come il numero del
                  menù */
  UWORD Qualifier;  /* è la copia del Qualifier dell'InputEvent */
  APTR IAddress;  /* contiene particolari indirizzi per funzioni intuition,
                     quale il puntatore al gadget selezionato */
  WORD MouseX, MouseY;  /* quando c'è un evento di riporto del movimento del
                           mouse, le cordinate sono memorizzate in queste
                           variabili; le cordinate sono relative all'angolo
                           in alto a sinistra della finestra */
  ULONG Seconds, Micros;  /* contengono l'ora dell'orologio di sistema in cui
                             si è verificato l'evento; Micros rappresenta i
                             microsecondi, Seconds i secondi */
  struct Window *IDCMPWindow;  /* puntatore della finestra su cui è stato
                                  generato l'evento */
  struct IntuiMessage *SpecialLink; /* variabile di utilizzo del sistema */
};
```

1.34 Il rinfresco delle finestre

Il Rinfresco delle finestre

Come voi tutti sapete le finestre possono essere spostate, ingrandite, rimpicciolite, fatte scivolare una davanti all'altra ecc.; il sistema di Amiga è progettato in maniera così flessibile che tutto ciò è gestito in maniera trasparente all'applicazione; vale a dire che l'applicazione disegna all'interno della finestra sempre nella stessa maniera e con le stesse cordinate, indipendentemente dalla posizione o grandezza di quest'ultima; un problemma tuttavia permane: cosa succede quando una zona oscurata da un'altra finestra viene riportata alla luce? In tal caso il sistema informa l'applicazione, mediante un messaggio IDCMP_REFRESHWINDOW alla porta IDCMP della finestra; l'applicazione una volta ricevuto questo messaggio, deve effettuare una chiamate BeginRefresh(), procedere con i dovuti rinfreschi al contenuto della finestra e chiamare EndRefresh(); nel caso non si debba effettuare alcun rinfresco, bisogna comunque chiamare la coppia delle funzioni appena viste senza alcuna istruzione al loro interno; sintassi di queste due funzioni è:

BeginRefresh(finestra);

dove "finestra" è il puntatore alla finestra di cui effettuare il rinfresco;

EndRefresh(finestra,finito);

"finestra" è la stessa della BeginRefresh corrispondente, "finito" è una variabile booleana e vale TRUE se il rinfresco è stato ultimato; FALSE se il rinfresco non è finito, allora il sistema mantiene bloccato il layer della finestra fino a quando non si esegue una coppia BeginRefresh/EndRefresh con "finito" impostato a TRUE; vi chiederete perchè spezzettare l'operazione di rinfresco in diversi blocchi: durante il rinfresco il sistema attende, per cui è IMPORTANTE che il codice del blocco di rinfresco sai molto veloce; spezzettando l'operazione di rinfresco in più sezioni, è agevole ripassare il controllo al sistema senza bloccarlo per un tempo esagerato.
Se non desiderate effettuare alcun rinfresco e quindi chiamare le due istruzioni appena indicate, oppure volete far realizzare il rinfresco totalmente al sistema, potete impostare il flag WA_NoCareRefresh.
Esistono tre tipi differenti di rinfresco organizzati dal sistema, ognuno con livello di efficienza e anche di consumo crescente:

Simple refresh
Simple refresh è il rinfresco più semplice: vengono mantenuti e rinfrescati tutti quei pixels che sono visualizzati sullo schermo; ad esempio, se una finestra viene spostata, il suo contenuto viene ricopiato; i problemi si verificano quando un'altra finestra passa davanti a quella "rinfrescata" in quanto il suo contenuto viene oscurato, per cui se la finestra viene spostata o riportata in avanti, la zona oscurata non viene rinfrescata.
Questo tipo di rinfresco si attiva impostando WFLG_SIMPLE_REFRESH nel tag WA_Flags.

Smart refresh
Nella finestra Smart refresh il sistema mantiene e rinfresca tutto il contenuto della finestra; nel caso visto prima, anche se la finestra con questo tipo di rinfresco viene oscurata, quando viene riportata alla luce la zona prima non visibile, viene correttamente ridisegnata; questo rinfresco si attiva impostando WFLG_SMART_REFRESH nel tag WA_Flags.

SuperBitMap refresh
Nelle finestre SuperBitMap l'utente specifica il BitMap (struttura che contiene le informazione per un'immagine o uno spazio grafico) che il sistema utilizzerà per il rinfresco della finestra; questo tipo di finestra serve quando quest'ultima può essere più piccola dell'immagine (nel qual caso la finestra mostra solo una porzione dell'immagine); ad esempio se si utilizza una finestra ridimensionabile, si disegna qualcosa e la si riduce, quando si riallarga la zona dell'immagine oscurata dal rimpicciolimento della finestra viene rinfrescata; naturalmente questo tipo di finestra si propone anche per altri utilizzi, quali ad esempio lo scorrimento di un'immagine più grande nella stessa, come fa MultiView; normalmente una finestra SuperBitMap è anche GimmeZeroZero, in modo da avere due layer differenti, uno per il contenuto e uno per il bordo; questo tipo di finestra si imposta con WFLG_SUPER_BITMAP in WA_Flags.

Mentre per i primi due rinfreschi serve semplicemente impostare il flag indicato ed il resto sarà compito del sistema, nella finestra SuperBitMap occorre allocare ed inizializzare una struttura BitMap nostra e passarla al tag WA_SuperBitMap all'apertura della finestra; per inizializzare una BitMap occorre allocare una struttura BitMap, inizializzarne i campi con la funzione:

```
InitBitMap(bitmap,profondità,ampiezza,altezza);
```

dove "bitmap" è il puntatore alla struttura BitMap da inizializzare, "profondità", "ampiezza" e "altezza" sono LONG indicanti la dimensione della bitmap; una volta inizializzata la bitmap, occorre allocare la memoria che contiene i piani dei bit dell'immagine mediante la funzione:

```
puntpiano = AllocRaster(ampiezza,altezza);
```

bisogna chiamare questa funzione per ogni bitplane da allocare, il cui puntatore è restituito in "puntpiano" che è di tipo PLANEPTR; questi puntatori dovranno essere inseriti nel campo vettore Planes della struttura BitMap; all'uscita del programma bisogna deallocare i bitplanes con

```
FreeRaster(puntpiano,ampiezza,altezza);
```

1.35 I gadgets

I gadgets

Nel mezzo del cammin del nostro corso su Intuition siamo giunti ad uno degli argomenti più importanti: i gadget. I bottoni, come vengono solitmante tradotti, sono dei pulsanti sullo schermo: l'utente infatti posizionandosi su di essi (identificati normalmente come rettangoli con aspetto 3D) e cliccando con il mouse attiva un'operazione, proprio come un pulsante; esistono due diverse categorie di gadget: di sistema e dell'applicazione; i primi li abbiamo già visti e utilizzati, sono infatti quelli già definiti dal sistema (gadget di profondità, di cambiamento di grandezza, di chiusura ecc.), i secondi sono definiti ed usati interamente dall'applicazione. I gadget possono essere divisi anche in quattro tipi, a seconda del funzionamento e delle caratteristiche:

Gadget

booleani
(o bottoni)
Sono gadget che funzionano semplicemente cliccandovi sopra e che una volta selezionati eseguono un'operazione.

Gadget
proporzionali
Questo tipo di gadget sono come delle manopole che permettono di indicare
uno fra diversi possibili valori; ad esempio gli slider per posizionare l'immagine nel MultiView.

Gadget
stringa
Sono gadget che, venendo selezionati dall'utente, permettono di inserire un
testo.

Gadget custom
I gadget custom sono una forma più generale di gadget che permettono di realizzare diversi tipi di funzioni e quindi risultano più flessibili di quelli visti.

Ogni gadget viene attivato quando l'utente clicca all'interno di una precisa regione specificata dall'applicazione, denominata select box; quando il gadget viene selezionato la sua immagine cambia per indicare all'utente di essere stato attivato; questa operazione viene denominata highlighting; un gadget può essere abilitato (quindi utilizzabile dall'utente) o disabilitato (quindi non utilizzabile); la disabilitazione viene indicata all'utente mediante il mascheramento del gadget con un pattern a griglia.
Un gadget di applicazione viene creato dichiarando ed inizializzando la struttura Gadget definita in "intuition/intuition.h"; Intuition avvisa l'applicazione sulla seleziona da parte dell'utente di un suo gadget mediante
IDCMP_GADGETUDOWN
e
IDCMP_GADGETUP
; il primo messaggio viene inviato appena l'utente seleziona con il pulsante sinistro del mouse il gadget, il secondo viene invece inviato quando l'utente rilascia il pulsante sul gadget; ambedue i messaggi hanno memorizzato nel campo
IAddress
l'indirizzo del gadget che è stato attivato.
I gadget appaiono sempre in una
finestra
o in un requester; possono essere creati all'apertura della finestra, o essere inseriti e rimossi successivamente; possono essere posti in un qualsiasi punto della finestra, anche sui bordi; la finestra, all'apertura, modifica automaticamente la grandezza dei bordi per contenere interamente i gadget inseriti su essi; la grandezza di un gadget può essere fissa o relativa alla grandezza della finestra; la posizione di un gadget è normalmente relativa per la cordinata x al bordo sinistro e per la cordinata y al bordo superiore della finestra, ma può essere relativa anche agli altri bordi.
Per creare una serie di gadget da agganciare alla finestra occorre definire e inizializzare una serie di strutture
Gadget

```
                , facendo bene attenzione di
prole in lista mediante il puntatore
                NextGadget
                , esempio:

struct Gadget MioGad1 = { NULL, ... };
struct Gadget MioGad2 = { &MioGad1, ... };
struct Gadget MioGad3 = { &MioGad2, ... };

In realtà non risulta necessario porli in lista se si desidera inserirli o
rimuoverli nella finestra una volta aperta, individualmente (vale a dire uno
per volta) mediante AddGadget() e RemoveGadget(); se si desidera però
inserirli tutti insieme o all'apertura della finestra o successivamente,
occorre metterli in lista così come mostrato e far riferimento al primo
elemento di quest'ultima che, nel nostro caso è MioGadg3 (attenzione è
MioGad3 che punta a MioGad2 il quale punta a sua volta a MioGad1 che
interrompe la lista con NULL); per inserirli all'apertura della finestra
bisogna porre il puntatore al primo gadget della lista nel tag
                WA_Gadget
                 in
OpenWindowTagList; se desiderate inserirli o rimuoverli successivamente
all'apertura della finestra bisogna utilizzare la funzione

pos = AddGList(finestra,gadget,posizione,numgad,requester);

dove "finestra" è il puntatore alla finestra in cui inserire i gadgets;
"gadget" è il puntatore al primo gadget della lista; "posizione" è la
posizione nella lista generale dei gadgets della finestra o del requester,
in cui inserire i gadget (con ~0 si inseriscono i gadgets alla fine della
lista); "numgad" è il numero di gadgets da inserire; "requester" è il
puntatore al requester in cui inserire i gadgets (se non è utilizzato
"finestra"); il valore ritornato "pos" indica la posizione in cui i gadgets
sono stati inseriti nella lista generale.
Per rimuivere una lista di gadgets occorre utilizzare la funzione

pos = RemoveGList(finestra,gadget,numgad);

dove i parametri assumono gli stessi significati descritti in AddGList()
eccetto che per "pos" che indica la posizione del primo gadget della lista
che è stato rimosso (vale -1 se il gadget non è stato trovato).
Non bisogna mai modificare gli attributi dei gadgets mentre sono inseriti
nella finestra; bisogna rimuoverli, modificarli e poi reinserirli con le
funzioni appena viste.

possa riconoscerlo e attivarlo; intuition permette di associare ad un gadget
una serie di informazioni grafiche in diversi modi; il programmatore può
decidere se utilizzare come immagine del gadget, un'immagine bitmap; in
questo caso bisogna inizializzare una struttura Image con i bitmap
dell'immagine, inserirne l'indirizzo nel campo GadgetRender della struttura
Gadget del gadget da creare, e impostare il flag
                GFLG_GADGIMAGE
                 nel campo
Flags sempre della struttura Gadget. Un'alternativa all'utilizzo
dell'immagine bitmap consiste nell'utilizzo di un bordo; con bordo si
intende una serie di linee che possono avere diverso colore; solitamente
questa tecnica viene utilizzata per creare i "bordi" del gadget e per
utilizzarla bisogna inizializzare una struttura Border con i dati delle
```

linee, passare il puntatore della struttura nel campo GadgetRender del gadget e non impostare il flag

GFLG_GADGIMAGE

(da questo si capisce che i bordi e le immagini non possono essere utilizzati insieme). Ad ogni gadget può essere associato insieme ad una delle due forme grafiche appena viste, un testo; per far questo bisogna inizializzare la struttura IntuiText con il testo e passarne l'indirizzo nel campo

GadgetText

del gadget. Il programmatore può decidere anche di non associare alcun grafico o testo al gadget, perché l'applicazione possiede già informazioni grafiche sufficienti o perché potrebbe non averne bisogno; un classico esempio consiste in un editor di testi, dove viene utilizzato un unico gadget che copra tutta la finestra testo; in questo modo l'applicazione viene avvisata quando l'utente clicca nella finestra testo e può agire di conseguenza (riposizionare il cursore o selezionare un blocco di testo).
Normalmente quando l'utente seleziona un gadget, questo cambia la propria immagine per indicare di essere in fase di selezione (fase di Highlighting); ci sono diversi modi per realizzare la fase grafica di selezione:

- Evidenziazione mediante complemento del colore
- Evidenziazione disegnando un box
- Evidenziazione mediante un'altra immagine o bordo
- Nessuna evidenziazione

L'evidenziazione mediante il complemento del colore, consiste nell'invertire bit a bit il codice dei colori del rettangolo di selezione; questa tecnica si ottiene attivanno il flag

GFLG_GADGHCOMP

nel campo Flags e risulta molto semplice da utilizzare; se i colori vengono impostati oculatamente l'effetto grafico può risultare qualitativamente elevato.
L'evidenziazione mediante il disegno di un box, consiste nel far disegnare ad intuition un rettangolo che contorni il gadget; questa evidenziazione si attiva impostando il flag

GFLG_GADGHBOX

nel campo Flags.
L'evidenziazione mediante immagine o bordo consiste nell'utilizzare un'altra immagine (se il disegno del gadget era costituito da un'immagine) o un altro bordo (se il rendering del gadget era un bordo) che indichino l'immagine selezionata; ovviamente bisogna inizializzare le relative strutture (Image nel primo caso e Border nel secondo) e passare il relativo puntatore nel campo SelectRender del gadget; bisogna anche specificare il flag

GFLG_GADGHIMAG
E nel campo Flags.

Grandezza e posizione del gadget

La posizione e le dimensioni del box di selezione del gadget sono definite dai campi LeftEdge, TopEdge, Width e Height del gadget, che indicano la posizione in pixels relativa ai bordi della finestra, e l'ampiezza e altezza sempre in pixels. L'immagine ed il testo del gadget sono sempre relativi alla posizione del gadget, per cui spostando quest'ultimo anche la sua immagine lo seguirà; la grandezza dell'immagine non è relativa alla grandezza del gadget, per cui se la dimensione del gadget viene modificata, bisogna effettuare il cambiamento nell'immagine oppure utilizzare un gadget BOOPSI.
Come sopra accennato, la posizione è relativa ai bordi della finestra;

normalmente LeftEdge è relativo al bordo sinistro (quindi si utilizzano valori positivi) e TopEdge è relativo al bordo superiore (si utilizzano valori positivi); vi è però la possibilità di rendere la posizione relativa verso gli altri bordi (un gadget che fa comunemente uso di ciò è quello di cambiamento della grandezza della finestra, che si trova sempre in basso a destra); se si vuole rendere LeftEdge relativo al bordo di destra (quindi si usano normalmente valori negativi) bisogna impostare il flag

GFLG_RELRIGHT
nel campo Flags; per rendere TopEdge relativo al bordo inferiore (si utilizzano valori negativi) si deve impostare il flag

GFLG_RELBOTTOM
nel campo Flags. Normalmente le dimensioni sono fisse, ma possono essere rese relative alla grandezza della finestra; bisogna impostare il flag
GFLG_RELWIDTH
per rendere l'ampiezza del gadget relativa all'ampiezza della finestra e
GFLG_RELHEIGHT
per fare lo stesso con l'altezza (possono anche essere usati singolarmente); il sistema per calcolare l'ampiezza (o altezza) del gadget, consiste nel sommare il valore del campo Width (o Height) del gadget con il valore dell'ampiezza (o altezza) della finestra; in tal caso si utilizzano normalmente valori negativi.
Si può decidere di posizionare i gadget sui bordi della finestra; in questa evenienza il sistema dimensionerà opportunamente la grandezza del bordo, per contenere tutti i gadget presenti sullo stesso; per indicare ad Intuition quali gadget si trovano sui bordi e di quali bordi si tratta, bisogna impostare i flags
GACT_RIGHTBORDER
(il gadget è nel bordo destro),

GACT_LEFTBORDER
(bordo sinistro),
GACT_TOPBORDER
(bordo superiore),

GACT_BOTTOMBORDER
(bordo inferiore) nel campo Activation del gadget.
Attenzione che Intuition adatterà la grandezza dei bordi solo all'apertura della finestra; per cui i gadget per i bordi devono gia essere tutti presenti nella lista dei gadgets passata ad
OpenWindowTagList
e se venissero aggiunti o rimossi successivamente, Intuition non modificherà la grandezza dei bordi. Alcune volte può capitare che il bordo non copra a livello grafico tutti i gadgets sui bordi, per ovviare a questo inconveniente basta creare un gadget "fantasma" (vale a dire senza immagine e che non serve a nulla) posto sul bordo e inserirlo in fondo alla lista dei gadget e, modificandone posizione e grandezza si potrà controllare la dimensione del bordo; si consiglia di rendere relativi al bordo destro i gadget presenti su tale bordo, e di rendere relativi al bordo inferiore i gadget presenti sullo stesso bordo; i gadget potrebbo essere presenti su due bordi contemporaneamente (se il gadget è in un angolo della finestra) per cui possono essere attivati due

GACT_xxxBORDER
adiacenti.
Come in ogni altro mezzo di comunicazione di Intuition anche i Gadget indicano all'applicazione di essere stati attivati mediante messaggi IDCMP; possiamo avere due comunicazioni, una indicante che l'utente ha selezionato (ma non rilasciato) il gadget (
IDCMP_GADGETDOWN
) e l'altra in cui l'utente lo ha rilasciato (
IDCMP_GADGETUP
); oltre ad indicare ad Intuition quali dei due tipi (o tutte e due) di messaggi ricevere, bisogna indicare per ogni singolo gadget quale azione deve compiere quest'ultimo una volta selezionato; impostando il flag
GACT_IMMEDIATE
nel campo Activation del gadget, Intuition invierà un
IDCMP_GADGETDOWN
all'applicazione quando l'utente selezionerà il gadget; impostando il flag
GACT_RELVERIFY
sempre nel campo Activation, Intuition invierà il messaggio
IDCMP_GADGETUP
all'applicazione quando l'utente rilascierà il gadget; se l'utente ↩
muove il puntatore al di fuori del gadget (mentre è attivato) e rilascia il pulsante, Intuition invierà un messaggio di tipo
IDCMP_MOUSEBUTTONS
. Per comprendere quale gadget ha generato il messaggio, basta controllare il campo IAddress di
IntuiMessage
, in cui è contenuto l'indirizzo della struttura Gadget del gadget che ci interessa; una volta ottenuto tale puntatore, si controlla il contenuto del campo GadgetID del gadget che contiene il codice identificatore di quest'ultimo, anche se basterebbe il semplice indirizzo (attenzione, prelevare il GadgetID solo quando si è certi che il messaggio è di tipo
IDCMP_GADGETDOWN
o
IDCMP_GADGETUP
).
Vi è la possibilità di essere informati sullo spostamento del mouse, mentre un gadget è selezionato; questo può risultare utile in caso di gadget proporzionali o in altri casi particolari; per essere ratificati di spostamenti del mouse, bisogna attivare il flag
GACT_FOLLOWMOUSE
nel campo Activation del gadget; Intuition invierà messaggi di tipo
IDCMP_MOUSEMOVE
ad ogni movimento del mouse (in questo tipo di messaggi non viene memorizzato il codice del gadget, quindi non lo si deve prelevare); nel caso di gadget booleani bisogna sempre specificare
GACT_RELVERIFY
per essere informati dei movimenti del mouse.

Attivazione e disattivazione dei gadgets

Un gadget può essere disattivato (quindi non utilizzabile dall'utente) con effetto a video indicato da un pattern a scacchiera che si sovrappone all'immagine del gadget; l'applicazione può indicare di aprire dei gadget già deselezionati, mediante

GFLG_DISABLED

; durante tutto il periodo di attività non possono essere attivati o disattivati agendo su questo flag, ma devono essere utilizzate le seguenti funzioni:

```
OnGadget(gadget, finestra, requester);
OffGadget(gadget, finestra, requester);
```

Dove gadget è il puntatore alla struttura gadget del gadget da abilitare o disabilitare; finestra è il puntatore alla finestra a cui il gadget è agganciato; requester è il puntatore al requester se il gadget è invece agganciato ad un requester.

I gadget

booleani

Torniamo a prlare di gadgets booleani, in realtà avete già tutti ↩ gli elementi per crearli ma mancano ancora alcuni aspetti; come già detto i gadget booleani sono come dei bottoni che implementano operazioni del tipo ON/OFF o Si/No; selezionandone uno questo funziona come un pulsante che attiverà un'operazione; vi è la possibilità di utilizzarli come "interruttori" cioè come indicatori di uno stato attivo/disattivo impostando

GACT_TOGGLESELECT

nel campo Activation della struttura Gadget, ma questo è stato già ↩ visto.

La caratteristica non ancora spiegata è il mascheramento (naturalmente non quello carnevalesco); normalmente il gadget ha forma rettangolare e le operazioni di selezione e di illuminazione (highlighting) vengono impostate su questa zona rettangolare; può accadere di volere forme diverse da quella rettangolare (come quella ovale) e per questo è stato realizzato il mascheramento; nel mascheramento il programmatore mette a disposizione di intuition una maschera (immagine di un bitplane della stessa grandezza del gadget) dove i bit ad 1 indicano i pixels effettivamente appartenenti al gadget; il mascheramento funziona per la selezione (se l'utente clicca sul gadget in un punto in cui la maschera è 0, allora intuition non selezionerà il gadget) e per l'illuminazione a complemento (il complemento dei bit dell'immagine verrà realizzato solo per quelli in cui la maschera ha 1), il mascheramento non funziona però per l'immagine; questo vuol dire che nel rinfresco dell'immagine verrà ridisegnato tutto il rettangolo andando incontro in alcuni casi ad effetti non voluti (se piazziamo il gadget ovale su un'immagine si osserveranno anche i quattro angoli del box del gadget). Per attivare il mascheramento bisogna inizializzare una struttura BoolInfo, il cui puntatore dovrà essere inserito nel campo

SpecialInfo

del gadget, e si dovrà attivare il flag

GACT_BOOLEXTEND

nel campo

Activation

sempre del

```
gadget.
Altra applicazione che potreste trovarvi davanti è la mutua
esclusione dei gadgets; vale a dire una serie di gadget di tipo
ToggleSelect raggruppati insieme, in cui la selezione di uno comporta la
deselezione di tutti gli altri (necessario quando bisogna selezionare uno
solo di una serie di serie di stati); questo tipo di applicazione non è
gestita da intuition, per cui il programmatore dovrà arrangiarsi; ma non vi
preoccupate perché arriva Zorro III il programmatore mascherato che vi
salva e vi spiegherà come fare! I passi per implementare la mutua
esclusione dei gadgets sono i seguenti:
- i gadgets vanno creati di tipo ToggleSelect (impostando il flag

                GACT_TOGGLESELECT
                 nel campo Action), con attivazione del solo tipo

                GACT_IMMEDIATE
                 (non utilizzare quindi
                GACT_RELVERIFY
                ), in tal maniera si
  gestiscono solo eventi
                IDCMP_GADGETDOWN
                - per l'illuminazione dei gadgets utilizzate una delle tecniche  ↩
                   già viste
- al verificarsi di un evento
                IDCMP_GADGETDOWN
                , dovranno essere rimossi i
  gadgets del blocco con un RemoveGList(), dovranno venir aggiustati gli
  stati di selezione mediante
                GFLF_SELECTED
                , quindi riagganciare i gadgets
  alla finestra mediante AddGList() e rivisualizzarli con RefreshGList().

I gadget
                proporzionali
                Spieghiamo ora come realizzare i gadget proporzionali; come già  ↩
                   descritto la
volta scorsa i gadget proporzionali permettono di specificare un valore in
un possibile range, funzionano un po' come le manopole a slitta del volume
di una radio; i gadget proporzionali sono costituiti da un contenitore (la
cui grandezza corrisponde a quella del gadget) e da una manopola (knob) che
scorre all'interno del contenitore; cliccando sulla manopola è possibile
spostarsi fluidamente da un punto all'altro del range dei possibili valori;
cliccando fuori dalla manopola ma sempre nel contenitore, si implementano
spostamenti in blocco; i gadget proporzionali possono muoversi sull'asse x,
y o entrambi e i movimenti hanno un limite teorico di 65536 posizioni, ma
sono sempre limitati alla risoluzione dello schermo.
Le applicazioni "logiche" dei gadget proporzionali sono sostanzialmente 2; la
prima (denominata scroller) in cui si mostra normalmente un numero limitato
di informazioni rispetto ad una grande quantità di dati; pensate ad esempio
ad un text-editor, in tal caso la manopola rappresenta la posizione e la
grandezza della "finestra" in cui vengono visualizzate le informazioni; in
questo tipo di applicazione è importante anche la grandezza della manopola,
perché rende l'idea della proporzione fra i dati visualizzati e quelli
effettivamente disponibili. La seconda applicazione (denominata slider)
impiega i gadget proporzionali per scegliere uno tra una serie di possibili
valori (pensate alla manopola per il volume o per la scelte di una
componente del colore).
```

I due casi sono sostanzialmente differenti dal punto di vista del programmatore, nel primo infatti la manopola rappresenta un insieme di dati ma permette di spostarsi ugualmente di uno in uno, nel seconodo invece la manopola seleziona un solo elemento di una serie; tutti e due i casi sono realizzabili dai gadget proporzionali di Intuition, anzi se notate bene il secondo è un caso particolare del primo (basta impostare il numero di elementi visualizzati ad 1).
Per realizzare un gadget proporzionale bisogna inizializzare una struttura ausiliaria, PropInfo il cui puntatore va inserito nel campo SpecialInfo del gadget; il primo campo da inizializzare è Flags, in cui è possibile specificare se non disegnare il bordo del gadget (
PROPBORDERLESS
, viene visualizzata solo la manopola); l'applicazione può specificare una propria immagine per la manopola o farne utilizzare una standard da Intuition (
AUTOKNOB
, l'immagine standard della manopola varia la propria grandezza in funzione di
HorizBody
e
VertBody
quindi risulta molto efficiente); bisogna comunque fornire una struttura Image anche se specificato
AUTOKNOB
(anche se in questo caso non c'è bisogno di inizializzare la struttura perché viene totalmente gestita da Intuition); la struttura Image non può essere condivisa da altri gadget proporzionali. Bisogna indicare in che direzione si può muovere la manopola, orizzontalmente (FREEHORIZ) e/o verticalmente (FREEVERT); anche i gadget proporzionali come tutto il resto del sistema ha cambiato il suo look in uno 3D molto più accattivante, per attivarlo (fatelo sempre) utilizzare il flag
PROPNEWLOOK
.
Altre variabili del PropInfo da inizializzare sono
HorizPot
e
VertPot
che indicano la posizione della manopola; questi valori vanno da 0 (posizione minima) a MAXPOT-body (posizione massima), dove body è la grandezza della manopola (che equivale al numero di elementi visualizzati) per cui occorre effettuare una proporzione per calcolare il giusto valore:

```
pot = ((val - minval) * MAXPOT) / ((maxval - minval) - numval);
```

dove minval è il valore minimo che la manopola può assumere, maxval è il suo valore massimo, numval è il numero di elementi visualizzati; val è il valore attualmente assunto dalla manopola e pot è il valore da inserire in HorizPot e/o VertPot; vediamo un esempio per fissare meglio l'idea: immaginiamo che la manopola debba specificare la componente Rosso (range 0 - 255) con un valore iniziale di 170:

```
UWORD pot;
struct PropInfo PI;
  .
```

```
  .
pot = ((170 - 0) * MAXPOT) / ((255 - 0) - 1);
PI.HorizPot = pot; /* e, oppure */
PI.VertPot = pot;
```

La conversione va fatta naturalmente anche quando bisogna leggere il valore della posizione in HorizPot o VertPot per sapere quale elemento (o gruppo) è stato selezionato con il gadget; la formula è la seguente:

```
val = (pot * ((maxval - minval) - 1)) / MAXPOT + minval;
```

dove pot è il valore prelevato da HorizPot o VertPot e le altre variabili assumono lo stesso significato prima descritto; verifichiamo l'esempio precedente:

```
UBYTE Red;
struct PropInfo PI;
  .
  .
Red = (PI.Horiz.Pot * ((255 - 0) - 1)) / MAXPOT + 0;
```

Gli altri due campi da inizializzare ed usare sono HorizBody e VertBody che indicano l'ampiezza e l'altezza della manopola; se è stato specificato AUTOKNOB nel campo Flags, l'immagine della manopola cambierà la propria dimensione in base a questi campi; come i Pot anche i Body variano da 0 a MAXBODY, per cui bisogna utilizzare delle proporzioni simili a quelle viste:

```
body = (numval * MAXBODY) / (maxval - minval);
```

dove numval è il numero di elementi o range visualizzato.
Le formule prima citate valgono sia per gli scrollers che per gli sliders, bisogna fare attenzione che nel primo caso "val" (l'elemento puntato dalla manopola) indichi il primo elemento visualizzato nella pagina, mentre per gli sliders occorre utilizzare come numval 1.
Per creare un gadget proporzionale occorre specificare
GTYP_PROPGADGET
nel
campo
GadgetType
della struttura Gadget e inserire il puntatore alla
struttura PropInfo nel campo
SpecialInfo
della struttura Gadget; ripetieamo
che quando si utilizza
AUTOKNOB
, bisogna far attenzione ad inserire
l'indirizzo di una struttura Image creata (non c'è bisogno di inizializzarla e gestirla, in quanto questo sarà compito di Intuition) nel campo GadgetRender della struttura Gadget (attenzione a non impostare il flag

GFLG_GADGIMAGE
). Per utilizzare una propria immagine basta semplicemente non specificare AUTOKNOB nel PropInfo ed indicare
GFLG_GADGIMAGE
nel Gadget; se
si vuole inoltre utilizzare l'immagine per l'illuminazione o selezione del gadget bisogna impostare il flag

GFLG_GADGHIMAGE
e fornire il puntatore alla
struttura Image (che deve avere stesse dimensioni di quella per l'immagine
non selezionata).
Per modificare i valori (flag, posizione e grandezza della manopola) senza
bisogno di staccare il gadget dal sistema, si utilizza la funzione
NewModifyProp():

```
NewModifyProp(gadget,finestra,requester,flags,horizpot,vertpot,horizbody,vertbody, ↩
    numgad);
```

dove gadget è il gadget proporzionale di cui cambiare i valori, finestra è il
puntatore alla finestra in cui il gadget è inserito, requester è il puntatore
al requester se invece il gadget è inserito in un requester, flags, horizpot,
vertpot, horizbody, vertbody sono i nuovi valori del
PropInfo
; numgad è il
codice di posizione in cui reinserire il gadget dopo che è stato deinserito
e aggiornato dalla funzione; si può impostare questo valore a tutti 1 (dato
che è un long quindi il valore è 0xFFFFFFFF) e in questa maniera, la
funzione aggiornerà le parti dell'immagine che sono effettivamente modificate,
ciò risulta notevolmente più veloce quindi è consigliabile.

I gadget
stringa
I gadget stringa permettono di inserire una stringa di caratteri; ↩
una
volta selezionato il gadget, intuition visualizzerà il cursore per inserire
nuovi caratteri; il cursore può essere spostato con i tasti cursore e con
l'intervento del mouse; quando l'utente preme il tasto RETURN o ENTER
oppure seleziona da qualche altra parte, allora il gadget è deselezionato.
Quando l'utente preme RETURN o ENTER viene generato un messaggio di tipo

IDCMP_GADGETUP
se era specificato
GACT_RELVERIF
Y; se il gadget viene
deselezionato cliccando altrove, allora non viene generato nessun messaggio

IDCMP_GADGETUP
. Il gadget può permettere di inserire anche solo un valore
numerico a 32 bit segnato(gadget stringa Integer); per impostare il gadget
stringa di tipo Integer, bisogna impostare il flag
GACT_LONGINT
nel campo
Activation del gadget; il valore numerico inserito è presente nel campo
LongInt della struttura StringInfo associata al gadget stringa.
E' possibile attivare il gadget stringa da applicazione mediante la funzione:

```
BOOL ActivateGadget(gadget, finestra, requester);
```

dove gadget è il gadget stringa da attivare, finestra è il puntatore alla
finestra a cui appartiene e requester è il puntatore al requester se il gadget
è inserito in un requester. Non è detto che l'operazione possa avere successo;
questo perché nel frattempo il sistema potrebbe essere impegnato con un menù o
con altre operazioni; se l'operazione di attivazione ha successo, allora

```
ActivateGadget() ritorna TRUE e FALSE altrimenti.
Vi è la possibilità dalla V37 del sistema, di passare da un gadget
stringa ad un altro premendo TAB o SHIFT+TAB; per abilitare il vostro
gadget stringa ad entrare in questo ciclo di selezione, occorre inserire
il flag
                GFLG_TABCYCLE
                ; l'ordine di selezione sarà determinato dalla
posizione dei gadgets nella lista; se un gadget viene deselezionato
con un TAB, Intuition invierà un messaggio
                IDCMP_GADGETUP
                 all'applicazione
(come se si fosse premuto RETURN o ENTER); per capire che la deselezione è
dovuta alla pressione del tasto TAB, si può controllare il campo Code del
messaggio che conterrà il valore 0x09 (il codice ASCII del TAB). Per
realizzare un gadget stringa, bisogna inizializzare una struttura

                StringInfo
                , passarne l'indirizzo nel campo
                SpecialInfo
                 del gadget e
specificare il flag
                GTYP_STRGADGET
                 nel campo
                GadgetType
                ; l'applicazione può
indicare l'allineamento della stringa, mediante
                GACT_STRINGLEFT
                ,

                GACT_STRINGCENTER
                 o
                GACT_STRINGRIGHT
                . Per il rendering del gadget stringa,
si può utilizzare uno qualsiasi dei metodi visti, mentre per
l'illuminazione si deve specificare necessariamente il complemento
(
                GFLG_GADGHCOMP
                ).
```

1.36 Struttura BoolInfo

```
Struttura BoolInfo

struct BoolInfo
{
  UWORD Flags;
  UWORD *Mask;
  ULONG Reserved;
};

Flags
Flags deve essere impostato con BOOLMASK

Mask
Mask è il puntatore ad un blocco di word contenenti i bit della maschera
```

```
(ved. listato)

Reserved
Reserved va impostato a NULL
```

1.37 Flag di attivazione del gadget

```
                Flag di attivazione del gadget

Questi flags sono utilizzati nel campo Activation del gadget

GACT_TOGGLESELECT
Questo flag può essere applicato solo ai gadget booleani; indica ad Intuitio
che il gadget è di tipo selezionato/deselezionato; si preimposta lo stato del
gadget con
                GFLG_SELECTED
                GACT_IMMEDIATE
Se questo flag è impostato, Intuition avvisa l'applicazione con

                IDCMP_GADGETDOWN
                 appena l'utente clicca sul gadget

GACT_RELVERIFY
Se questo flag è impostato, Intuition avvisa l'applicazione con
IDCMP_GADGETUP, quando l'utente rilascia il gadget; se l'utente lascia il
pulsante del mouse al di fuori del box di selezione del gadget, Intuition
invierà un normale messaggio
                IDCMP_MOUSEBUTTONS
                GACT_ENDGADGET
Utilizzato solo in gadget da utilizzare in requester; se questo flag è
impostato, quando l'utente seleziona tale gadget si ottiene lo stesso effetto
di una chiamata EndRequest() (il requester viene chiuso)

GACT_FOLLOWMOUSE
Una volta selezionato il gadget, Intuition riferirà di ogni movimento del
mouse con
                IDCMP_MOUSEMOVE
                , fino a quando l'utente non rilascerà il pulsante
del mouse; in caso di gadget booleano bisogna impostare anche
                GACT_RELVERIFY
                ,
altrimenti Intuition non avviserà sugli spostamenti del mouse

GACT_RIGHTBORDER
Il gadget va posto nel bordo destro della finestra; la grandezza del bordo
verrà adattato per contenere il gadget

GACT_LEFTBORDER
Il gadget va posto nel bordo sinistro della finestra; la grandezza del bordo
verrà adattato per contenere il gadget

GACT_TOPBORDER
Il gadget va posto nel bordo superiore della finestra; la grandezza del bordo
verrà adattato per contenere il gadget
```

```
GACT_BOTTOMBORDER
Il gadget va posto nel bordo inferiore della finestra; la grandezza del bordo
verrà adattato per contenere il gadget

GACT_STRINGCENTER
Se il gadget è di tipo stringa, il testo inserito sarà posto al centro del
gadget

GACT_STRINGRIGHT
Se il gadget è di tipo stringa, il testo inserito sarà posto a destra nel
gadget

GACT_STRINGLEFT
Se il gadget è di tipo stringa, il testo inserito sarà posto a sinistra nel
gadget

GACT_LONGINT
Se il gadget è di tipo stringa, permette di inserire solo un valore numerico
a 32 bit con segno

GACT_ALTKEYMAP
Indica di utilizzare un KeyMap alternativo per questo string gadget; bisogna
impostare il campo AltKeyMap nella struttura StringInfo relativa al gadget
stringa

GACT_BOOLEXTEND
Utilizzabile solo con gadget booleani; se impostato allora il gadget booleano
ha una struttura
                BoolInfo
                 associata; il puntatore a BoolInfo deve essere
inserito nel campo
                SpecialInfo
                 della struttura gadget

GACT_STRINGEXTEND
E' un flag obsoleto definito e utilizzato in V36; indica che il campo

                StringInfo
                .Extension punta ad una valida struttura StringExtend; questo flag
è rimpiazzato da
                GFLG_STRINGEXTEND
                 dalla V37
```

1.38 Flags dei gadget

```
                Flags del gadget

Questi flags vanno impostati nel campo
                Flags
                 del gadget:

flags per il tipo di selezione grafica, uno dei seguenti deve essere
selezionato:
GFLG_GADGHNONE
Nessuna evidenziazione
```

```
GFLG_GADGCOMP
Evidenziazione mediante complemento bit a bit

GFLG_GADGHBOX
Evidenziazione mediante rettangolo intorno al gadget

GFLG_GADGHIMAGE
Evidenziazione con immagine o bordo alternativo

i seguenti flags possono essere impostati insieme a quello per il tipo di
selezione grafica:
GFLG_GADGIMAGE
Il gadget ha un'immagine bitmap; se si utilizza un bordo state attenti a non
impostare questo flag

GFLG_RELBOTTOM
La posizione del gadget è relativo al bordo inferiore; per calcolare la
posizione effettiva del gadget, Intuition somma TopEdge alla cordinata del
bordo inferiore; per cui, volendo mantenere il gadget all'interno della
finestra bisogna utilizzare valori negativi per TopEdge; se il flag non è
impostato TopEdge è relativo al bordo superiore

GFLG_RELRIGHT
La posizione del gadget è relativo al bordo destro; per calcolare la
posizione effettiva del gadget, Intuition somma LeftEdge alla cordinata del
bordo destro; per cui, volendo mantenere il gadget all'interno della
finestra bisogna utilizzare valori negativi per LeftEdge; se il flag non è
impostato LeftEdge è relativo al bordo sinistro

GFLG_RELWIDTH
L'ampiezza del box di selezione del gadget è relativo all'ampiezza della
finestra; per calcolare l'ampiezza effettiva del gadget viene sommato Width
all'ampiezza della finestra, per cui occorre utilizzare valori negativi per
Width, se si vuole mantenere il gadget all'interno di quest'ultima

GFLG_RELHEIGHT
L'altezza del box di selezione del gadget è relativo all'altezza della
finestra; per calcolare l'altezza effettiva del gadget viene sommato Height
all'altezza della finestra, per cui occorre utilizzare valori negativi per
Height, se si vuole mantenere il gadget all'interno di quest'ultima

GFLG_SELECTED
Se questo flag è impostato, lo stato di partenza per un gadget booleano
seleziona/deseleziona (cliccando una volta si seleziona, cliccando una seconda
volta si deseleziona) è selezionato; altrimenti è deselezionato

GFLG_DISABLED
Se questo flag è attivato, il gadget è disabilitato; per abilitare o
disabilitare un gadget mentre è inserito nella finestra utilizzare le funzioni
OnGadget() e OffGadget()

GFLG_STRINGEXTEND
Il campo estensione di
                StringInfo
                 punta ad una struttura StringExtend; questo
flag è ignorato prima della versione V37 del sistema; per la versione V36
```

```
utilizzare il flag
                GACT_STRINGEXTEND
                GFLG_TABCYCLE
Se il gadget è di tipo stringa, viene permesso il passaggio da un gadget
stringa ad un altro mediante TAB o SHIFT-TAB
```

1.39 Tipo di gadget

```
                Tipo di gadget

Questo valore indica il tipo di gadget da creare e va impostato nel campo
GadgetType del gadget:

GTYP_BOOLGADGET
Gadget di tipo booleano (ON/OFF)

GTYP_STRGADGET
Gadget di tipo stringa; permette l'inserimento di una stringa di caratteri;
per costringere ad inserire un valore numerico intero, impostare il flag

                GACT_LONGINT
                 nel campo Activation

GTYP_PROPGADGET
Gadget di tipo proporzionale

GTYP_CUSTOMGADGET
Normalmente non impostato dall'applicazione; usato per gadget custom BOOPSI

I seguenti flag possono essere impostati in aggiunta ad uno di quelli
precedentemente elencati:

GTYP_GZZGADGET
Se il gadget deve essere posto sul bordo di una finestra
                GIMMEZEROZERO
                ,
bisogna impostare questo flag

GTYP_REQGADGET
Impostare questo flag se il gadget è posto in un requester
```

1.40 Struttura PropInfo

```
Struttura PropInfo

Riportiamo la struttura PropInfo con la descrizione dei suoi campi:
struct PropInfo
{
  UWORD Flags;
  UWORD HorizPot;
  UWORD VertPot;
  UWORD HorizBody;
```

```
  UWORD VertBody;
  UWORD CWidth;
  UWORD CHeight;
  UWORD HPotRes,VPotRes;
  UWORD LeftBorder;
  UWORD TopBorder;
};

Flags
I possibili flags del PropInfo sono:
- PROPBORDERLESS: crea il gadget proporzionale senza bordo.
- AUTOKNOB: se non è impostato l'applicazione dovrà occuparsi dell'immagine
  della manopola, altrimenti verrà realizzata automaticamente dal sistema.
- FREEHORIZ e FREEVERT: impostare FREEHORIZ se la manopola si muove
  orizzontalmente e FREEVERT se si muove verticalmente
- PROPNEWLOOK: utilizza il nuovo look 3D per il rendering del gadget
- KNOBHIT: viene impostato da Intuition quandi la monopola è selezionata
  dall'utente.

HorizPot e VertPot
Indicano la posizione della manopola (varia da 0 per la posizione minima a
MAXPOT per quella massima, per cui occorre effettuare una proporzione);
bisogna inizializzarli prima di inserire il gadget nella finestra; una volta
agganciato alla finestra, questi campi possono essere solo letti; per
modificarli bisogna prima rimuovere il gadget (RemoveGadget), scrivere i
campi e reinserire il gadget nella finestra (AddGadget), oppure utilizzare
la funzione NewModifyProp().

HorizBody e VertBody
Indicano la grandezza orizzontale e verticale della manopola; tali valori
variano da 0 a MAXBODY; impostare la grandezza a MAXBODY se non vi è alcun
elemento da visualizzare o se quelli esistenti sono inferiori alla capienza
della finestra (per gli altri casi occorre effettuare una proporzione per
determinare esattamente la grandezza della manopola); anche qui valgono gli
stessi discorsi sulla lettura o modifica sopra descritti.

Le restanti variabili sono utilizzate da Intuition.
```

1.41 Struttura StringInfo

```
Struttura StringInfo

Riportiamo e descriviamo la struttura StringInfo per i gadget stringa
struct StringInfo
{
  UBYTE *Buffer;
  UBYTE *UndoBuffer;
  WORD BufferPos;
  WORD MaxChars;
  WORD DispPos;
  WORD UndoPos;
  WORD NumChars;
  WORD DispCount;
  WORD CLeft, CTop;
  struct StringExtend *Extension;
```

```
  LONG LongInt;
  struct KeyMap *AltKeyMap;
}
```

Buffer
L'applicazione deve fornire un vettore di caratteri (stringa) in cui Intuition memorizza la stringa inserita dall'utente; il puntatore a tale vettore deve essere inserito in questo campo; la dimensione del vettore non deve essere inferiore a MaxChars; il testo presente nel vettore prima di inserire il gadget nella finestra, verrà visualizzato al momento dell'inserimento di quest'ultimo; se il gadget è di tipo Integer bisogna memorizzare la codifica ASCII del numero preinserito.

UndoBuffer
E' il puntatore al buffer di caratteri per l'operazione di undo (opzionale); l'operazione di undo permette di ripristinare il contenuto precedente alle modifiche; l'UndoBuffer può essere condiviso da più string gadget (dato che un solo string gadget alla volta può essere attivo), attenzione ad usare un numero di caratteri corrispondente alla massima esigenza fra tutti i string gadgets.

MaxChars
Indica il numero massimo di caratteri che la stringa può avere; questo valore comprende anche il NULL di fine stringa, quindi il numero effettivo di caratteri memorizzabili è MaxChars-1.

BufferPos
BufferPos è inizializzato con il valore della posizione del cursore nel buffer della stringa; questo valore varia da 0 alla lunghezza della stringa meno 1.

DispPos
DispPos indica il primo carattere visualizzato nel gadget; può accadere che se la stringa inserita è più lunga della capienza grafica del gadget, il testo viene fatto scorrere all'interno di quest'ultimo.

UndoPos, NumChars, DispCount, CLeft, CTop
Queste variabili sono mantenute da Intuition e non dovrebbero essere modificate da Intuition. UndoPos specifica la posizione del carattere nel buffer undo; NumChars specifica il numero di caratteri correntemente presenti nel buffer; DispCount specifica il numero di caratteri attualmente visualizzati nel gadget.

Extension
Permette di specificare un ulteriore struttura estensiva per ulteriori controlli sul gadget

LongInt
Contiene il valore intero inserito nel gadget se il gadget stringa è di tipo Integer.

AltKeyMap
Per default i caratteri sullo schermo appaiono mediante una semplice trasformazione in ASCII; volendo utilizzare una KeyMap alternativa bisogna specificare il puntatore alla struttura KeyMap relativa in questo campo e impostare GACT_ALTKEYMAP nel campo Activation del gadget.

1.42 Struttura Gadget

```
Struttura Gadget

Viene riportata e descritta la struttura Gadget per la definizione dei
gadgets:

struct Gadget
{
  struct Gadget *NextGadget;
  WORD LeftEdge, TopEdge;
  WORD Width, Height;
  UWORD Flags;
  UWORD Activation;
  UWORD GadgetType;
  APTR GadgetRender;
  APTR SelectRender;
  struct IntuiText *GadgetText;
  LONG MutualExclude;
  APTR SpecialInfo;
  UWORD GadgetID;
  APTR UserData;
};

NextGadget
Puntatore al prossimo gadget; tutti i gadgets di una finestra vengono
mantenuti in una lista

LeftEdge, TopEdge, Width, Height
Posizione e grandezza del select box del gadget

Flags
Flags per la descrizione di alcune caratteristiche del gadget

Activation
Tipo di attivazione del gadget

GadgetType
Tipo di gadget (BOOL, PROP ecc.)

GadgetRender
Puntatore a struttura Image per immagine o Border per bordo, indicante il
disegno o rendering del gadget

SelectRender
Dello stesso tipo di GadgetRender indicante però l'immagine o bordo del
gadget selezionato

MutualExclude
Doveva servire per la mutua esclusione fra gadgets, ma non è stato
implementato ed è quindi obsoleto; dalla V36 del sistema viene utilizzato
come puntatore all'hook per un gadget Custom

SpecialInfo
Puntatore ad una speciale struttura dati, per ulteriori informazioni in caso
di gadget proporzionali o stringa
```

GadgetID
Di uso libero per l'applicazione; normalmente utilizzato come codice identificatore del gadget

UserData
Puntatore a blocco dati dell'applicazione

1.43 I menù

I menù

I menù costituiscono una parte vitale del sistema, un mezzo importante per l'iterazione con l'utente; i menù (non ci riferiamo naturalmente a quelli del ristorante) servono ad indicare all'applicazione di compiere determinate azioni, o ad indicare determinati attributi (ad esempio grandezza, tipo e caratteristiche di un font da una lista); le stesse operazioni possono essere svolte da una finestra o un requester con una serie di gadgets, ma il gran vantaggio dei menù risiede nel fatto che sono a scomparsa, vale a dire che le opzioni non vengono normalmente mostrate sullo schermo, ma vengono visualizzate solo con la pressione del stato destro del mouse. Una volta premuto il tasto destro, apparirà al posto della barra titolo dello schermo una prima lista di opzioni indicanti i titoli dei menù; una volta selezionato uno di questi titoli, apparirà un box con una lista di elementi; selezionando uno di questi e rilasciando il pulsante del mouse, si indica all'applicazione di aver selezionato l'opzione indicata; l'opzione può però possedere a sua volta una sotto lista, in tal caso selezionando l'opzione apparirà affianco un'altra lista dando la possibiltà di selezionare un opzione ad un livello successivo; gli elementi che indicano gli attribuiti vengono contraddistinti con un checkmark (di solito un'immagine a forma di v) se selezionati. Ogni finestra può avere un suo menù anche se vengono visualizzati nello stesso punto (se le finestre sono presenti sullo stesso schermo), dato che solo una finestra può essere quella attiva. Gli elementi dei menù sono costituiti a livello grafico da un testo, da un'immagine o da tutti e due insieme. La principale limitazione dei menù risiede nel numero di opzioni e di livelli utilizzabili; il numero di livelli utilizzabili è 3 (titoli menù, menù e sottomenù), ogni livello può possedere al massimo rispettivamente 31, 63 e 31 elementi.
Per attivare un menù occorre utilizzare la funzione di Intuition:

```
ris = SetMenuStrip(finestra,menu);
```

dove finestra è il puntatore alla struttura
Window
della finestra
in cui inserire il menù; menu è il puntatore alla struttura
Menu
contenente tutte le definizioni dei menù (vedremo fra breve come è strutturato); il valore ritornato "ris", è un BOOL che indica successo o fallimento; in realtà non può accadere che il menù non venga installato per cui la funzione ritorna sempre TRUE; la funzione invece per rimuovere un menù è:

```
ClearMenuStrip(finestra);
```

le due funzioni vanno chiamate all'inizio e all'uscita del programma
(oppure nel momento in cui si decide di visualizzare e rimuovere il menù),
ma comunque sempre dopo aver aperto la finestra e prima di chiuderla
rispettivamente. Se si decide in un particolare momento, di cambiare la
struttura del menù (opzioni visualizzate, disabilitate ecc.) occorre prima
rimuovere il menù, effettuare le modifiche sulla relativa struttura, e
riagganciare il menù alla finestra una volta effettuate tutte le modifiche;
c'è la possibilità però di non usare necessariamente SetMenuStrip, se le
modifiche effettuate sono solo l'abilitazione/disabilitazione di opzioni o la
selezione/deselezione di attributi con checkmark; in tal caso la funzione
utilizzabile per agganciare un menù (attenzione utilizzare solo se in
passato è stata chiamata la funzione SetMenuStrip), che risulta essere
molto più "leggera" è:

```
ris = ResetMenuStrip(finestra,menu);
```

dove i parametri passati e ricevuti ricoprono lo stesso significato prima
descritto.
Per disabilitare totalmente il menù (infatti anche quando nessun
menù e agganciato, premendo il tasto destro appare la barra del menù),
occorre specificare il flag

```
WA_RMBTRAP
 nel tag
WA_Flags
 di
```

OpenWindowTagList(); questo flag non solo impedisce di visualizzare la
barra titolo, ma permette anche di individuare la pressione del tasto
destro, inviando un messaggio

```
IDCMP_MOUSEBUTTONS
```

. Una stessa struttura menù
può essere condivisa ed utilizzata da più finestre, a condizione che le
finestre siano presenti sullo stesso schermo; tenete presente che così
facendo lo stesso menù apparirà per diverse finestre, a meno di non
cambiarne la struttura all'attivazione di ognuna di queste
(

```
IDCMP_ACTIVEWINDOW
```

); l'applicazione in tal caso deve anche attivare

```
IDCMP_MENUVERIFY
```

, in modo da impedire che l'utente utilizzi il menù di una
finestra appena attivata, prima che l'applicazione abbia gestito l'evento

```
IDCMP_ACTIVEWINDOW
 relativo.
```

Una volta che l'utente rilascia un menù (anche senza aver selezionato alcuna
opzione) il sistema invierà un messaggio di tipo

```
IDCMP_MENUPICK
```

; il codice
dell'opzione eventualmente selezionata è presente nel campo Code della
struttura IntuiMessage del messaggio di Intuition; il codice è di 16 bit ed
è così suddiviso: i primi 5 bit (0-4) indicano il numero del menù, gli altri
6 bits (5-10) indicano il codice dell'opzione del menù e i restanti 5 bits
(11-15) indicano il codice dell'opzione di un eventuale sotto-menù. Nel caso
nessun opzione sia stata selezionata il valore Code, equivale ad una
costante denominata MENUNULL; per prelevare i codici del menù, opzione e
opzione del sotto-menù non è necessario andare ad esaminare direttamente i
bit di Code, basta semplicemente utilizzare le seguenti macro sempre

definite in "intuition/intuition.h": MENUNUM(Code) restituisce il numero del menù, ITEMNUM(Code) restituisce il numero dell'opzione, SUBNUM(Code) restituisce il numero dell'oipzione del sotto-menù; i valori restitui partono da 0 (primo menù o opzione), 1 (secondo menu' o opzione) e arrivano fino 30 o 62 (31-esima o 63-esima opzione o menù).
Come ben sapete utilizzando quotidianamente Amiga, è possibile effettuare la multi-selezione nei menù; ciò significa che mentre gironzolo per i menù mantenendo premuto il tasto destro, posso selezionare più opzioni premendo il tasto sinistro; il sistema genererà in tal caso sempre un solo messaggio

IDCMP_MENUPICK
, ma come individuare tutte le opzioni selezionate allora? Il codice ritornato si riferisce ad uno degli elementi selezionati, nel campo NextSelect della struttura MenuItem dell'opzione relativa, s itroverà il codice di un'altra opzione selezionata, e così via fino a quando il campo NextSelect non vale MENUNULL; rimane a questo punto il problema di come determinare l'indirizzo di una struttura
MenuItem
partendo dal codice; il sistema ci viene in soccorso con la seguente funzione:

```
Opzione = ItemAddress(menu,codicemenu);
```

dove menu è il puntatore alla struttura
Menu
del menù, codice è il codice dell'opzione selezionata e Opzione è il puntatore alla struttura
MenuItem
dell'opzione relativa. Dopo quanto detto il ciclo di programma per l'esaminazione delle selezioni per i menù è il seguente:

```
struct IntuiMessage *msg;
struct Menu *menu;
UWORD CodiceMenu;
struct MenuItem *Opzione;
  .
  .
CodiceMenu = msg -> Code;
while (CodiceMenu != MENUNULL)
{
  Opzione = ItemAddress(menu,CodiceMenu);

  /* processo l'attuale opzione */

  CodiceMenu = Opzione -> NextSelect;
}
```

E' possibile dalla V37 del sistema far indicare se l'utente desidera avere delle informazioni sull'azione compiuta dall'opzione del menù; infatti l'utente mentre seleziona un'opzione del menù, può premere il tasto HELP; in tal caso il sistema si comporterà come se l'utente avesse effettivamente selezionato ma, invece di inviare un messaggio di tipo
IDCMP_MENUPICK
, ne invierà uno di tipo
IDCMP_MENUHELP
, e verrà passato nel campo Code il

codice dell'opzione relativa; in questo caso l'applicazione (ciò non è ovviamente obbligatorio) dovrà visualizzare le informazioni di aiuto per quella opzione.
Nell'inizializzazione delle strutture
Menu
e
MenuItem
, occorre
indicare il box di selezione dell'opzione o del titolo del menù; bisogna far attenzione nel considerare la grandezza del font quando si decide la grandezza del box e inoltre, bisogna stare attenti (questo per versioni del sistema prima della V37) che i box di opzioni successive siano adiacenti; il box contenente le opzioni del menù verrà automaticamente dimensionato dal sistema ed il box del sotto-menù può sovrapporsi al box del menù; esiste in realtà un modo più semplice per realizzare menù e gadget ed è la gadtools.library che vedremo in una delle prossime puntate.
Come abbiamo già detto le opzioni possono essere di tipo attributo; ciò vuol dire che un'opzione può essere selezionata o deselezionata indicando così uno stato (attivo o disattivo); per specificare che un'opzione è di tipo attributo bisogna indicare il flag CHECKIT nel campo
Flags
della
relativa struttura MenuItem; se l'opzione è selezionata, verrà attivato il flag CHECKED sempre nel campo Flags. Normalmente quando si sceglie un'opzione attributo questa viene sempre selezionata, indipendentemente dallo stato in cui si trova; se si specifica il flag MENUTOGGLE verrà invertito lo stato attuale; è possibile effettuare la mutua-esclusione fra opzioni attributo, in maniera che selezionando un'opzione è possibile deselezionarne altre; ciò è reso possibile dal campo
MutualExclud
e; questo
è un campo di tipo LONG, in cui ogni bit rappresenta un'opzione (bit 0 prima opzione, bit 1 seconda opzione ecc.); al momento della selezione, le opzioni i cui rispettivi bit sono impostati ad uno verranno deselezionate.
Vediamo un esempio per comprendere meglio questo meccanismo, supponiamo infatti di dover specificare mediante menù lo stile di un testo, i campi con i relativi valori di mutual-exclude saranno:

```
Plain     0xFFFE
Bold      0x0001
Italic    0x0001
Underline 0x0001
```

in tale maniera se seleziono Plain, resetterò tutte le altre opzioni fuorché se stessa (infatti tutti i bit con posizione maggiore di 0 sono impostati ad 1, mentre il bit 0 a 0); se seleziono Bold, Italic o Underline disattiverò solamente la prima opzione (Plain) in quanto solo il bit 0 è impostato a 1.

1.44 Le strutture per i menù

Le strutture per i menù

Le strutture utilizzate per definire un menù sono due: Menu e MenuItem; la prima serve per definire i titoli dei menù, la seconda serve per definire un'opzione del menù.

struct Menu

```
{
  struct Menu *NextMenu;
  WORD LeftEdge, TopEdge;
  WORD Width, Height;
  UWORD Flags;
  BYTE *MenuName;
  struct MenuItem *FirstItem;
  WORD JazzX, JazzY, BeatX, BeatY;
};
```

NextMenu
Puntatore al prossimo titolo del menù.

LeftEdge, TopEdge, Width, Height
Posizione e grandezza del box di selezione del titolo del menù.

Flags
- MENUENABLED
  se presente indica che questo menù è abilitato, altrimenti risulta disabilitato; in tal caso le opzioni e le sotto-opzioni del menù saranno mascherati e l'utente non potrà selezionarli.
- MIDRAWN
  questo flag indica che il menù è correntemente visualizzato all'utente.

MenuName
Puntatore alla stringa che verrà visualizzata come titolo del menù.

FirstItem
Questo punta alla struttura MenuItem indicante la prima opzione di questo menù.

JazzX, JazzY, BeatX, BeatY
Valori utilizzati solo dal sistema

```
struct MenuItem
{
  struct MenuItem *NextItem;
  WORD LeftEdge, TopEdge;
  WORD Width, Height;
  UWORD Flags;
  LONG MutualExclude;
  APTR ItemFill;
  APTR SelectFill;
  BYTE Command;
  struct MenuItem *SubItem;
  UWORD NextSelect;
};
```

NextItem
Puntatore al prossimo MenuItem della lista delle opzioni.

LeftEdge, TopEdge, Width, Height
Posizione e grandezza del box di selezione dell'opzione.

Flags
- CHECKIT
  indica al sistema che questa opzione è un attributo e che deve visualizzare

```
  il checkmark se il flag CHECKED è impostato
- CHECKED
  se CHECKIT è impostato, indica che questa opzione attributo è selezionata
- ITEMTEXT
  se questo flag è impostato, indica che i campi ItemFill e SelectFill
  contengono puntatori a strutture IntuiText per il rendering dell'opzione;
  altrimenti contengono puntatori a strutture Image
- COMMSEQ
  se questo flag è settatto, l'opzione contiene una scorciatoia
- MENUTOGGLE
  usato insieme a CHECKIT; se questo flag è impostato l'opzione può essere
  deselezionata ricliccandola
- ITEMENABLED
  se impostato indica che l'opzione è abilitata alla selezione, altrimenti è
  disabilitata
- HIGHFLAGS
  i flag descritti qui di seguito, indicano il tipo di illuminazione:
  - HIGHCOMP
    si effettua il complemento di tutti i bits del box di selezione
  - HIGHBOX
    disegna un rettangolo attorno all'opzione
  - HIGHIMAGE
    isualizza un testo o un immagine alternativa (dipende da ITEMTEXT)
  - HIGHNONE
    nessuna illuminazione

i seguenti flags sono utilizzati solo da intuition:
- ISDRAWN
Intuition imposta questo flag quando il sotto-menù di questa opzione è
visualizzato
- HIGHITEM
Intuition imposta questo flag quando l'opzione è illuminata.

MutualExclude
Questo valore indica quale altre opzioni attrributo del menù devono essere
mutualmente escluse dalla selezione dell'opzione; ogni bit di questa LONG
rappresenta un'opzione; le opzioni mutualmente escludibili sono le prime 32

ItemFill
Puntatore ai dati utilizzati per il rendering dell'opzione; nel caso sia
settato il flag ITEMTEXT nel campo Flags questo campo punta ad una struttura
IntuiText, altrimenti punterà ad una struttura Image; attenzione, più
strutture IntuiText (o Image) linkate insieme mediante il campo Next....
verranno visualizzate insieme; questo permette di creare rendering complessi
anche utilizzando semplici testi.

SelectFill
Se il flag HIGHIMAGE è settato nel campo Flags, allora questo campo deve
possedere un puntatore a struttura IntuiText o Image per il rendering nella
fase di illuminazione; se specificato ITEMTEXT nel campo Flags, questo campo
deve puntatore ad una struttura IntuiText, altrimenti deve puntare ad una
struttura Image.

Command
Questo è un valore char che contiene il codice ASCII del tasto di
scorciatoia; tale valore sarà valido se specificato COMMSEQ nel campo Flags;
il tasto verrà visualizzato insieme all'immagine del tasto Amiga sulla
```

destra dell'opzione.

SubItem
Puntatore alla struttura MenuItem della prima opzione del sottomenù; non utilizzare questo campo nel caso l'opzione faccia già parte di un sotto-menù, in tal caso infatti il sistema ignorerà tale campo.

NextSelect
Questo campo è scritto da Intuition per indicare all'applicazione quali sono gli altri elementi selezionati dall'utente in caso di multi-selezione.