La programmazione del Mac
Le Risorse (3)

di Raffaello De Masi

Siamo giunti alla fine della trattazione delle risorse, i mattoni della programmazione Mac. Questo leviatano, nascosto nelle viscere del toolbox, spaventoso solo a nominarsi, si sta, spero, mostrando nella sua vera veste di agnellino disposto ad assecondare le richieste e le esigenze del programmatore come il più servizievole degli schiavi. È giunto, adesso, il momento di tirare le somme e di concludere il nostro dire, facendo effettivamente vedere come si fa a invocare una risorsa e ad usarla nella maniera più efficiente

Oggi vedremo come utilizzare le risorse sia come layer in sé, sia come tecnica di trasferimento di dati da un resource block. Come dice Chris Stazny in un suo bellissimo articolo dedicato, le risorse sono il quid che distingue il gentleman driver dall'autista da noleggio. Vediamo allora come creare una semplice risorsa e come maneggiarla adeguatamente, nella sua struttura, perché questa ci segua obbedientemente nei nostri scopi.

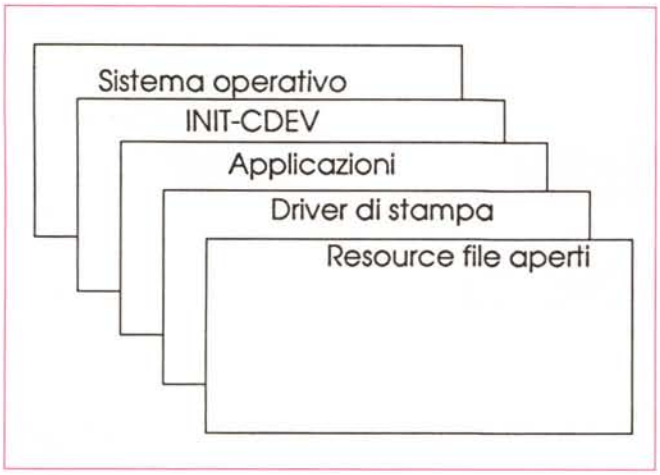
Partiamo dalla fine (Stazny dice che solo i ditch-digger, i minatori, cominciano dall'inizio); al momento dell'accensione il Mac apre automaticamente, in bootstrap, il System File. La resource map (ricordate la puntata scorsa) è letta in memoria e la lista completa delle risorse disponibili è allocata in un'area sicura e inaccessibile della memoria stessa per essere disponibile immediatamente per ogni circostanza (attenzione, è sempre disponibile la lista delle risorse, ma non le risorse stesse; come dicemmo l'altra volta, infatti, alcune di esse sono marcate come «purgeable», cancellabili, se ci fosse necessità di spazio libero in memoria; d'altro canto questo avviene anche in ogni altra applicazione gestita attraverso risorse). Non solo, ma pressoché tutti i programmi accedono a risorse del System stesso. Un esempio lampante è l'uso delle font, caricate nel System attraverso il Font DA Mover; ogni applicazione che gestisca caratteri accederà a queste risorse attraverso una chiamata a sistema operativo, senza avere la necessità di mantenere un (inutile) set personale di FOND, FONT e NFNT.

Permetteteci una piccola precisazione, che sembrerà ovvia, ma non è certo peregrina. Il System File è sempre (ovviamente) caricato prima e solo successivamente una applicazione è eseguita come uno startup-program. In altri termini, il resource file dell'applicazione è materialmente presente in memoria dopo la mappa di sistema. Questo è importante in quanto, al momento della invocazione della risorsa, l'ultima parte caricata è la prima soggetta a ricerca. Immaginiamo che l'applicazione contiene una risorsa del tipo ICON con ID number pari a 1, e il System File contiene anch'esso un ICON, differente nel contenuto, ma con lo stesso ID. Non avverrà mai che ci possa essere confusione in qualche modo visto che, nella ricerca, verrà incontrata prima la risorsa dell'applicazione poi quella del System (per la verità, la gestione del Multifinder ha complicato notevolmente la cosa, ma non è questo il momento di metterci da soli i bastoni tra le ruote).

Immaginiamo ora di voler accedere, per qualche nostro motivo programmatorio, alla risorsa ICON del sistema e non a quella della applicazione (e questo, ovviamente, senza chiudere l'applicazione stessa, o, addirittura chiamandole tutte e due). Bisogna in qualche modo «avvisare» il Resource Manager di bypassare il primo «appuntamento» fino a trovare la risorsa giusta al livello più basso. La cosa, come si può immaginare, non è certo semplice, e potrebbero, nella gestione del programma stesso, verificarsi errori dalle conseguenze inaspettate se l'indirizzamento non è quello giusto (immaginate se, appunto, in Multifinder fossero contemporaneamente caricati più di due o tre programmi).

La soluzione sta «nell'informare» il Resource Manager di usare il System File come se fosse il resource file corrente; così ogni file portato in memoria dopo il System sarà sempre subordinato al System stesso

Per conoscere quale è il resource file



La struttura a strato della mappa di risorsa.

corrente qualsiasi linguaggio di programmazione mette a disposizione chiamate adeguate. In ZBasic avremo:

```
curRes - FN CURRESFILE
CALL USERESFILE(0)
'
' aggancio al resource file di sistema
'
CALL USERESFILE(curRes)
```

In pratica questa sovrapposizione di risorse e la relativa gestione è un poco più complessa. Utility di startup, come CDEV e INIT portano i loro set di risorse durante la fase di inizializzazione alla accensione; e così file di risorse diverse, ma potremmo dire simili, possono essere contemporaneamente presenti a confondere le idee. Ad esempio ZBasic e MSBasic caricano il printer driver al top della applicazione corrente e, ancora, giusto per complicare ulteriormente le cose, file di risorse, agganciati tramite le chiamate OPENRESFILE o OPENRFPERM, sono sistemati al vertice della pila e immediatamente aperti.

Come fare, in questo caso, a dipanare la matassa ed a navigare nel porto sicuro della risorsa che in quel momento ci serve? Occorre tener conto che i resource file sono identificati sempre da un numero di riferimento. E attraverso questo numero è possibile riconoscere e indirizzare le diverse risorse, proprio perché questi numeri hanno caratteristiche distintive diverse a seconda dell'applicazione che li contiene. Il System File può essere sempre raggiunto con un resource number iniziante con zero; inoltre l'applicazione corrente ha un res_number aggiornato alla locazione di memoria &900 e il toolbox provvede, in silenzio e in maniera del tutto trasparente, a fornire (e aggiornare) un numero di riferimento ad ogni file che viene aperto nell'applicazione.

In questo modo è possibile guardare a una resource map dello stato attuale della memoria in maniera più ordinata e precisa; come si vede in figura la memoria sarà, in un certo modo stratificata (non a caso layer significa, appunto strato), con punti d'accesso codificati e gestiti, appunto dal toolbox, attraverso il resource manager.

Precauzioni nella gestione del res_manager

Occorre, come al solito nella vita, avere un poco di pazienza nel lanciarsi nell'uso «spregiudicato» di questa nuova tecnica. Il resource manager mette a disposizione una serie numerosa di comandi, che mettono praticamente a disposizione del programmatore tutti i comandi del toolbox. Ma attenzione! La chiamata a certe procedure, peraltro potenti, se non è fatta con criterio e a conoscenza completa delle funzionalità della stessa, può portare, per imperizia del programmatore, a spiacevoli conseguenze.

Un esempio è la procedura ADDRESRESOURCE, che aggiunge una risorsa al resource file corrente. Nella illustrazione allegata, probabilmente, questa chiamata sarebbe raffigurabile come un oggetto sistemato al top di ogni stack.

Essa si usa in questo modo:

— se si lavora interattivamente all'interno di un ambiente di programmazione, è necessario aggiungere risorse al resource file aperto. Questo viene ottenuto attraverso il comando

```
resRef - FN OPENRESFILE
("Nome_del_file")
```

— si sta operando in una applicazione stand-alone; in questo caso si può aggiungere la risorsa direttamente nel resource fork dell'applicazione. Questo si ottiene, secondo quanto già detto in precedenza, con la chiamata

```
PEEK WORD(&900)
```

Una volta scelta la tecnica operativa, occorre salvare il numero di riferimento della risorsa presente alla sommità dell'heap. Passare allora al file su cui si sta lavorando e invocare ADDRESOURCE. Passare al file originale. A questo punto verrebbe logico aggiungere direttamente al file la risorsa, ma non è così: se, per caso, nel file esiste una vecchia conversione della risorsa, con lo stesso ID number, la nuova non sostituirà automaticamente la vecchia, ma verrà solo aggiunta. Avremo così un duplicato che interferirà con il corretto funzionamento del sistema.

La soluzione è abbastanza semplice, e bene a conoscenza di chi ha lavorato con il toolbox in maniere avanzata. Creare un handle per la risorsa con una chiamata a FN GETRESOURCE. Se l'handle è diversa da zero, la risorsa esiste e va preventivamentre rimossa con RMVERESOURCE. Una piccola precisazione. Un bug nella gestione degli handle comporta che la rimozione di una risorsa dal resource map non cancella automaticamente l'handle relativo dalla lista tenuta dal memory manager. Occorre allora fare una chiamata supplementare a FN DISPOSHANDLE, per assicurarsi la eliminazione della handle dalla lista.

I passi totali del processo saranno, riassumendo:

— chiamata al resource file corrente;
— passaggio al nostro resource file;
— controllo se e dove esiste una risorsa analoga;
— cancellazione della risorsa esuberante;
— aggiunta della nuova risorsa;
— passaggio al file originale.

Ricuperiamo i valori

Bene, abbiamo fatto un bel passo avanti, ma non è ovviamente tutto. Occorre tirare fuori le informazioni dalle risorse. La maniera più semplice per farlo è recuperare i valori attraverso le variabili.

Ovviamente la prima cosa da fare è il dimensionamento delle variabili, destinate ad accogliere i dati in memoria. Un caso piuttosto diffuso è quello della chiamata alla stampante. Dopo le operazioni di Dimensionamento (che, ad esempio, il driver di stampa esegue automaticamente) useremo una chiamata a BLOCKMOVE per determinare quanto sarà grande il blocco dati da collegare

all'handle che gestirà il trasferimento dei dati (ricordate lo scrapfile che Macwrite crea e poi cancella?).

Un esempio di procedura in tal senso potrebbe essere:

```
DIM prefPort
DIM prefBaud
DIM prefParity
DIM prefWordSize
DIM prefStopBits
DIM PrefBuffer&
'

preRecSize = VARPTR(prefRecSize -
VARPTR(prefPort)
```

A questo punto la cosa diviene più semplice. Passando il blocco dati alla risorsa un semplice frammento di codice potrebbe essere:

```
ID - 100
tp& = VCI("Prfs")
hndl& = FN GETRESOURCE(tp&,ID)
BLOCKMOVE PEEK(hndl&) - VARPTR
(prefPort),prefRecSize
```

Resta ancora da salvare i dati nella risorsa. La cosa avviene in maniera abbastanza simile a quella precedente. Dopo aver settato il resource file corrente, e cancellato (non dimenticheremo mai di ricordarlo) eventuali informazioni su vecchie preferenze o risorse, creeremo un handle di grandezza sufficiente a contenere i dati. La fase successiva sarà muovere le variabili in questo handle e completare il processo aggiornando il resource manager alla nuova situazione. Il codice potrebbe essere:

```
hndl& = FN NEWHANDLE(prefRecSize)
BLOCKMOVE VARPTR(prefPort),-PEEK
LONG(hndl&),prefRecSize,
CALL ADDRESOURCE
```

Attenzione, comunque che, creando un handle si crea automaticamente anche un blocco fluttuante di memoria non direttamente riferibile a una risorsa. Attenzione: anche il semplice trasferimento di informazioni in variabili DIMensionate non significa aver creato una risorsa. Ricordare infine che la chiamata a ADDRESSOURCE esegue diverse operazioni:

— l'handle è agganciata a resource map, e il blocco diviene a tutti gli effetti una risorsa;
— la risorsa è marcata come modificata ed è manipolata direttamente dal resource manager;
— il resource manager informa il memory manager che esiste una nuova handle, attivata e funzionante, che ha bisogno di una notifica prima di essere spostata o cancellata.

E così abbiamo concluso con le risorse: dalle prossime volte questa rubrica potrebbe cambiare forma, e in parte, anche contenuti, continueremo ad interessarci di programmazione, ma affronteremo anche altri aspetti poco noti del sistema operativo interno del Mac; specialmente adesso che il System 7 ci ha dato la possibilità di sbizzarrirci in campi del tutto nuovi e inconsueti. A presto.

MC