RISC-OS: overview sul Kernel (2)

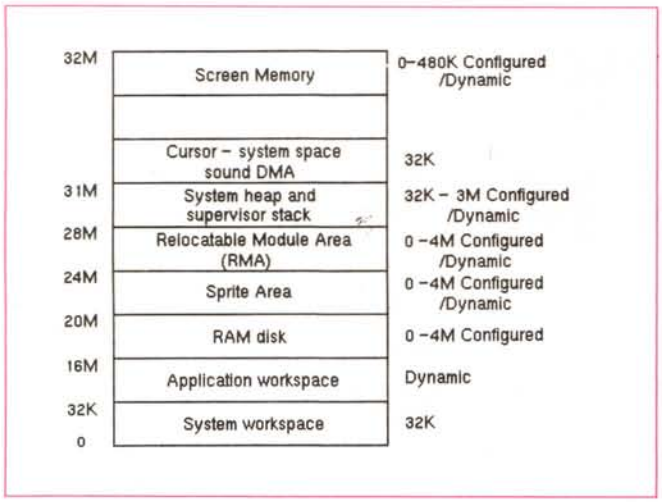
di Bruno Rosati

Seconda «dispensa» di RISC-OS e secondo appuntamento con il Kernel. Una overview che conclude il quadro generale dell'atomo kerneliano, eccezion fatta per il VDU driver la cui trattazione verrà compiutamente affrontata nel prossimo incontro

Differentemente da quanto avevo annunciato al momento di chiudere l'articolo del mese scorso — in cui davo appuntamento per una «conferenza» sulla Video Display Unit — rivedendo un po' la logica di trattazione che mi sono imposto, ho trovato più esatto riservare alla VDU stessa il prossimo articolo anziché il presente. In effetti si tratta di un'argomentazione piuttosto complessa che oltre a riguardare il discorso Kernel fa importanti riferimenti alla interazione tra la VDU ed un primo gruppo di moduli d'estensione. Parlare oggi di Video Display e riferirsi ai moduli del System Estension, senza aver concluso la trattazione di tutti i rimanenti sistemi presenti nel Kernel, oltre a rappresentare un modo un poco disordinato di procedere, potrebbe tra l'altro creare una certa confusione. A conferma di ciò, da questa seconda «overview Kerneliana» è esclusa anche l'analisi relativa al sistema degli Sprite, proprio perché, la gestione degli stessi più l'interazione con gli Estension Module (Font Manager, Draw module, Colour-Trans e Window Manager) forma in pratica un argomento unico che s'incentra sulle valenze e le caratteristiche dell'Unità Video. Riservando tutto un articolo a tale tema, il prossimo mese oltre a concludere l'effettiva fase dedicata allo studio delle peculiarità del Kernel, in pratica iniziere-

Tabella di riferimento, per nome e valore, per la custumizzazione del formato di stampa del Time and Date.

Name	Value	Example
CS	Centi-seconds	99
SE	Seconds	59
MI	Minuts	05
12	Hours in 12 hour formats	07
24	Hours in 24 hour formats	23
AM	«AM» or «PM»	PM
PM	«AM» or «PM»	AM
WE	Weekend, in full	Thursday
W3	Weekend, in three char.	Thu
WN	Weekend, as a number	5
DY	Day of the month	01
ST	«st», «nd», «rd», or «th»	st
MO	Month name, in full	September
M3	Month name, in three char	Sep
MN	Month as a number	09
CE	Century	19
YR	Year within century	87
WK	Week of the year	52
DN	Day of the year	364
0	Insert an ASCII 0byte	
%	Insert a «%»	



Mappa della Memoria Logica. Tale condizione è quella relativa all'accensione della macchina o subito dopo un reset.

Esempio di allocazione della memoria in un A-310 con paginazione da Size di 8K.

AREA	PAGES	PAGE SIZE	TOTAL
FontSize	20	4K	80K
RamFsSize	0	8K	0
RMASize	16	8K	128K
ScreenSize	20	8K	160K
SpriteSize	10	8K	80K
SystemSize	4	8K	32K +32K
System workspace			32K
Cursor etc. workspace			32K
Total	576K		
Application area	1024K–576K= 448K		

mo a parlare anche del System Estension Module.

Se prendete un attimo la tabella apparsa nell'articolo del mese scoso, dopo la trattazione dei sistemi per il Character Output ed il Character Input, esclusi il VDU e gli Sprite, questa seconda parte dell'overview concentra la sua attenzione sul restante gruppo di sistemi a livello di Kernel: ovvero quello relativo alle routine di conversione, i moduli rilocabili, il memory management, il time and Date e il Program Environment.

Time and date... e conversioni

Time and Date vuol dire clock, ovvero, un valore immagazzinato e modificato nel tempo con incrementi regolari. Vedi il movimento del mouse, la conta reale di un orologio con allarme — tipo quello disponibile nel «block-notes» presente a livello di desktop — in settaggio e la catalogazione del tempo di scrittura legato ad un determinato file, etc.

Tutto il processo di timerizzazione seguito sotto RISC-OS avviene con frazioni di un centesimo di secondo e viene delegato a quattro differenti tipi di timer: il cosiddetto *monotonic timer*, il *system timer*, l'*interval time* e il *real-time clock*. Il vero e proprio «orologio» è proprio l'ultimo timer citato, ovvero, come anche il nome lo evidenzia: il real timer. Un *clock* che s'immagazzina con un valore a 5 byte nei CMOS dell'orologio di bordo e che riflette un uso normale della datazione. Il classico tipo 00:00:00. Quello che può essere utilizzato (con settaggio a cura dell'utilizzatore) anche con l'Alarm o la Clock application del desktop. Allo stesso modo il RISC-OS utilizzerà il real time clock per datare un file e distinguerne la stesura cronologica. In generale tale tipo di timer può essere solo letto e sincronizzato; mai riscritto.

A «sola lettura» risulta anch'essere il *monotonic timer*. Immagazzinato con un valore distribuito su 4 byte esso incrementa normalmente ogni centesimo di secondo ed è predisposto al time-stamping relativo alle applicazioni. Il system clock a sua volta può esser invece alterato ed è particolarmente usato per la quantificazione del tempo trascorso dentro l'ambiente di lavoro di una determinata applicazione.

Così come il monotonico anche il system clock si resetta ad ogni reset di tipo hardware.

L'*interval timer* infine rientra nella importante categoria dei *timer event* ed è utilizzabile per creare gli stati di attesa — wait — facendo riprendere determinati processi elaborativi, dopo un certo tempo di pausa, nel momento in cui la sua conta si porterà nuovamente a zero. Se, per esempio, si abbisogna di un wait di 10 secondi (calcolando che anche l'*interval timer* incrementa ogni centesimo di secondo) si dovrà settare un valore negativo di «–1000». Quando il timer si riporterà a zero, l'esecuzione del programma in run riprenderà normalmente. Ovvero si sarà verificato un evento.

A livello di RISC-OS è anche possibile, attraverso l'uso di stringhe, disporre l'utilizzatore alla customizzazione delle modalità con le quali tempo e data vengono stampati. Se noi scriviamo ad esempio una stringa del tipo:

```
%W3,%DY                    %M3
      %CE%YR.%24:%MI:%SE
```

complicata solo apparentemente a livello sintattico, altro non faremo che obbligare il riordino del print del Time and Date. Il RISC-OS rivolgerà il nostro ordine alla SWI dedicata — ovvero la OS_ConvertDataAndTime — che una volta letto il «tempo» lo rinvierà ordinato come è nel comando impartito dall'user.

Di conseguenza, per primo vedremo stampare il nome del giorno della settimana, abbreviato nei primi tre caratteri che lo compongono, quindi il numero del giorno, poi quello del mese (anch'esso abbreviato) appresso l'anno ed infine l'ora, i minuti ed i secondi.

Se guardate la tabella pubblicata nella pagina precedente vi renderete conto che si tratta di una cosa estremamente comprensibile.

Altro importante argomento è quello relativo alle capacità di conversioni possibili in ambiente RISC-OS.

Invero «semplici» chiamate ad interrupt predisposte dal sistema per modificare, da una forma all'altra, un valore numerico, una stringa, un'espressione, l'argomento di una stringa in equivalente numerico, etc.

Senza entrare nei particolari tecnici, in RISC-OS è possibile convertire numeri in stringhe e stringhe in numeri, mutare le coppie di numeri (network e station) di una informazione ricevuta via Econet, quindi il taglio di un file in una stringa infine, come estrema elasticità operativa, la possibilità data dall'utente di poter trasformare il numero di una SWI nell'equivalente valore-stringa e viceversa.

Moduli rilocabili

Una delle caratteristiche più interessanti garantite dal Kernel archimediano, fin dai tempi del vecchio Arthur, è sicuramente quella della possibilità di usare i cosiddetti moduli rilocabili. Ovvero, parti di software elaborabili dall'utente che, una volta attivati, permettono una ulteriore estensione del sistema operativo.

La validità di tale opzione è estremamente evidente: potendolo (e volendolo) ciascun utente-programmatore è messo in grado di rendere il RISC-OS particolarmente dedicato alle proprie, specifiche esigenze. Integrandolo con propri sistemi o arrivando a sostituire i singoli moduli operativi.

Come ulteriore esemplificazione si pensi alla grossa opportunità di poter aggiungere nuove SWI o più potenti «*command».

I moduli rilocabili occupano un'area della memoria chiamata RMA (Relocatable Module Area) e sono aggettivati come «rilocabili» proprio perché, in quest'area dedicata, possono essere caricati in specifiche locazioni di memoria indirizzabili da parte del programmatore stesso.

Detto cosa sono, è interessante vedere anche quando è consigliabile (e talvolta persino necessario) usare i moduli.

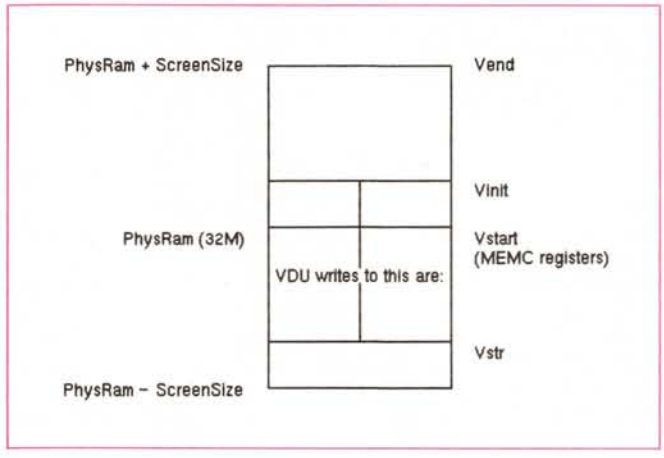
Il Programmer's Reference Manual al riguardo enuncia una sorta di «tavola dei comandamenti», dove il primo requisito attribuibile ai moduli è che, nella maggior parte dei casi, siano utilizzati per la realizzazione di routine estensive al Sistema. Tra l'altro se ne consiglia l'uso come «librerie mobili» di linguaggi ad alto livello come il «C».

Altra caratteristica basilare da osservare nella programmazione di un modulo dovrà esser quella della loro grandezza, sempre concentrata in pochi byte e il loro indirizzamento nelle zone della RAM residente in modo che siano sempre pronti alla chiamata e che a questa rispondano in modo rapido.

Una seconda serie di «comandamenti» riguarda le modalità d'uso dei moduli ed è centrata sulla valenza della SWI OS_Module, predisposta al caricamento, l'inizializzazione, il run e la rimozione di ogni modulo. Si tratta di una chiamata particolarmente efficace e, di conseguenza, sfruttabile dal programmatore, giacché tramite essa è posibile esaminare ed eventualmente cambiare l'ammontare dello spazio della RMA attribuibile ad un modulo.

Molto interessante è, per così dire, l'iter consigliato per le modalità di scrittura.

Screen Memory.



Un modulo, così spiega il Programmer's Reference Manual, ha il suo centro attivo nel *module header;* ovvero la tavola descrittiva composta da 11 entrate particolarmente finalizzata al dialogo con il RISC-OS al quale riservano tutte le informazioni relative al modulo ricreato.

La tavola contenente le informazioni riguardanti l'indirizzo di inizio dell'esecuzione del modulo, il codice di inizializzazione, il nome del modulo.

Gestione della memoria

Ovvero: il Memory Management.

Come il RISC-OS è in grado di gestire la memoria a disposizione di Archie lo abbiamo visto attraverso la descrizione del MEMC. Il chip delegato al controllo e alla distribuzione della memoria fisica ricostituita in equivalenti mappe logiche. Riprendendo da dove ci eravamo fermati la volta scorsa, riannodiamo il discorso introducendo il concetto di *management* che il RISC-OS opera sulla stessa mappa logica.

Il RISC-OS, come è noto, è in grado di automatizzare l'allocatura di determinati programmi — meglio dire utility — costituiti in grandezze estremamente contenute. Differentemente, molti programmi e, nel caso specifico della trattazione che si sta facendo: i moduli rilocabili, necessitano dell'assegnazione per così dire arbitraria di memoria. L'elasticità del sistema comunque garantisce l'eventuale riutilizzazione della stessa subito dopo l'uso.

A riguardo il PRM cita gli esempi inerenti il filing system ed alcuni driver del VDU (uno per tutti quello per il Font Manager).

Allocazioni e de-allocazioni di memoria sono azioni che un programmatore compie assai spesso nel suo lavoro. Il Memory Management predisposto sotto RISC-OS è in grado di offrire facilitazioni estremamente potenti. Una delle strutture più interessanti a riguardo di management è quella degli heap; «accumulatori» di memoria che una volta conformata la stessa in un blocco, permettono al programmatore di allocarne o liberarne una o più parti separate.

Gli heap possono stare dentro la RMA, nel caso siano connessi ad un modulo, oppure dentro la zona di lavoro di un programma. A riguardo, il RISC-OS contiene un vero e proprio Heap Management system predisposto all'allocazione delle zone di RMA. Senza comunque correre più oltre, torniamo nel seminato osservando la figura relativa alla Mappa della Memoria Logica. Scalando uno per uno i vari gradini del sistema di set-up, partiamo dai 32K permanentemente assegnati al workspace del sistema. Un'allocazione prefissata ed immutabile questa, a cui fan-

HANDLER*
Undefined instruction
Prefetch abort

Data abort
Address exception
Error
CallBack
BreakPoint
Escape

Event
Exit
Unused SWI
UpCall

*Attenzione: tutte le chiamate installate ad Handler
passano attraverso il ChangeEnvironment V.

Lista degli handler disponibili in RISC-OS.