L'Atari Multitasking

Beckemeyer Multiuser/multitasking C-Shell

di Paolo Ventafridda

Produttore
Beckemeyer Development Tools
478 Santa Clara Ave., Suite 300
Oakland, CA 94610
Tel. 001-415-452-1129
BBS: 001-415-452-4792

Distributore:
Nessun distributore in Italia.
Rivolgersi alla Beckemeyer Usa per i più vicini rivenditori europei.

Prezzi:

MT C-Shell system	*.. $129.95*
VSH Manager	*.. $ 34.95*
On-Line Manuals	*.. $ 19.95*
C-tools	*.. $ 24.95*
UUCP driver	*.. $ 45.95 (annunciato)*
Scheda 4 porte RS232	*.. $495.00*
Spedizione dagli Usa	*.. $ 20.00*

Configurazione:

Computer:	*Atari ST (520,1040 o Mega)*
Memoria RAM:	*minimo 520K (consigliata 1024K)*
Memoria di massa:	*almeno 2 disk-drive 360K oppure* *1 solo disk drive 720K oppure* *1 disk drive 360K con 1 Ram disk* *oppure: Hard disk (consigliato)*

Note:
La prova è stata fatta sulla versione completa della MT C-Shell, comprendente VSH (Visual Shell), manuali online, C-tools (tot.: $210).
La Beckemeyer supporta i propri utenti garantendo assistenza, upgrades, annunci nuove release.
Il Software non è protetto.

GEM. TOS & tasks

Uno degli aspetti maggiormente criticati del sistema operativo Atari è quello della limitazione al single-task. Nei «diverbi» con i possessori di Amiga, uno dei luoghi comuni era «...Amiga è multitasking, Atari no...».

Per molti la magica parola «multitasking» è sinonimo di potenza ed efficienza; a mio avviso su macchine come Atari e Amiga lo è fino ad un certo punto. Se è la velocità che cerchiamo, un ambiente multitasking è certamente penalizzante. Su micro di questo ordine, poi, crescono verticalmente i rischi di «crash» del sistema.

Comunque sia, se a qualcuno serve il multitasking su ST, si aprono due strade:
1) abbandonare il TOS (ed il GEM) per un O.S. differente, quale OS/9, Idris, Minix, Mirage ecc. ecc.
2) cercare di «patchare» il TOS implementando un Kernel e adottando un'interfaccia utente tipo Shell.

Entrambe le soluzioni hanno vantaggi e svantaggi: la prima permette di ottenere un sistema affidabile, ma costringe ad abbandonare il parco programmi esistente (i programmi TOS e GEM non potranno ovviamente funzionare); la seconda soluzione renderà il sistema «instabile», pur offrendo un limitato utilizzo in time-sharing delle risorse della macchina. È in quest'ottica che andiamo ad esaminare un prodotto come la *MT C-Shell*, realizzato dalla Beckemeyer Development Tools (BDT), negli U.S.A.

Il package era pubblicizzato su una rivista del settore come «multiuser/multitasking O.S. running GEM and TOS programs»: l'ho acquistato nel novembre 1987 e ve lo presentiamo in anteprima.

L'ambiente C-Shell

L'interfaccia utente della C-Shell (che d'ora in avanti abbrevieremo con MTC) è un interprete di comandi molto simile a quello sotto *BSD-Unix 4.3*: gli autori hanno realizzato una biblioteca di utility molto ricca, che spazia da comandi fondamentali come «grep» a meno noti quali «uniq»...

Lo stesso manuale consiglia la lettura di un qualunque testo su Unix per approfondire la conoscenza del sistema e la programmazione di shell scripts, peraltro solo accennata nella documentazione di base. Questo per sottolineare lo sforzo fatto per realizzare un prodotto più vicino possibile ad uno standard esistente (almeno in apparenza).

A differenza di altri Sistemi Operativi per Atari, quali l'*OS/9*, la *MT C-Shell è assolutamente compatibile con il TOS ed il GEM*. Significa che non dovrete formattare alcun disco per poterci scrivere sotto MTC, e che tutti i vostri programmi gireranno normalmente se lanciati dalla MTC.

Potrete cioè utilizzarla come «CLI» (comand language interpreter) alternativa al Desktop sotto GEM.

Quando i file si trovano all'interno di una lunga lista di subdirectory, l'accesso «via mouse» diventa lungo e scomodo: con la MTC sarà sufficiente scrivere il path name completo ed il nome del file, esattamente come in *MS-DOS*, tanto per fare un altro esempio.

Se lanciate un programma GEM, comparirà il desktop normalmente e lavorerete come se la MTC non esistesse. Al termine, ritornerete al prompt della Shell, che avrà ripreso il comando.

La coesistenza di MTC e GEM è senza dubbio segno di un grosso sforzo di programmazione da parte di David

Beckemeyer, autore di tutto il pacchetto.

La configurazione richiesta per poter lavorare con la MTC è quella media: *1 drive da 720K*, e almeno *1 mega RAM*. Secondo il manuale anche con soli 520K a disposizione si può installare una versione ridotta; in realtà ci si accorge ben presto che 1 Mega è appena sufficiente, figuriamoci la metà...

L'uso di un Hard Disk è consigliato, ma non indispensabile: un sistema di partenza potrebbe essere composto da due drive da 720K oppure uno da 720 e uno da 360; è possibile tuttavia utilizzare anche un RAM disk, nel caso la memoria abbondi (Mega-2 e Mega-4 ST).

Va data per scontata comunque la conoscenza dell'ambiente *UNIX*, o perlomeno un interesse ad acquisirne le basi. Se non avete la più pallida idea di cosa sia Unix e non avete né tempo né voglia di imparare qualcosa di nuovo, beh, la MTC non fa per voi.

Il boot

Esistono *3* differenti tipi di boot disponibili: *single-user, multi-user, visual shell*.

I primi due non richiedono la presenza del GEM, e quindi permettono una installazione immediata all'accensione della macchina. Naturalmente non sarà possibile in seguito utilizzare software di tipo .PRG: solo i .TOS verranno correttamente eseguiti.

Usando il boot insingle user (*init1.prg*), si accede alla shell come «*root*» (letteralmente: radice, su Unix indica lo stato di super-utente per funzioni di amministrazione del sistema). Il prompt di default è un cancelletto, non viene chiesta alcuna password: con *logout* o *exit* si ritorna in ambiente GEM automaticamente.

Lanciando *init2.prg* si inizializza il sistema in *multiuser* mode, e vengono lanciati due processi in più rispetto al single user. Uno è il daemon di stampa, che provvede a bufferizzare i testi per stamparli poi in background; l'altro è il *cron*, che ogni minuto controlla il proprio file */usr/lib/crontab* ed eventualmente esegue i programmi prestabiliti. Uno di questi è *atrun.prg*, che a sua volta controlla la directory */usr/spool/at/* per vedere se è stata ordinata l'esecuzione di qualcosa nell'ora e giorno corrente.

Al boot in multiuser inoltre viene controllato il file */etc/ttys*: se è stata configurata come attiva (enabled) la porta seriale, il *getty.prg* sarà eseguito anche su quel device. È quindi possibile collegarsi da remoto (via modem ad esempio) come terminali dell'Atari, usando programmi e risorse della macchina (entro certi limiti, che poi esamineremo).

In multiuser il sistema chiede di identificarsi (*login*), e non permette di accedere alla shell senza essere registrati (*accouting*): per uscire dalla MTC Shell occorrerà fare il logout e rientrare come utente «*shutdowns*», con password uguale a quella di root. Solo a questo punto si ritornerà al Desktop GEM. Il terzo tipo di boot è quello della *visual shell (VHS)*: in pratica, anziché abbandonare il GEM, si entra in un ambiente con i tipici pull-down menu dai quali si può sia lanciare programmi normalmente, sia «aprire» fino a 4 window di sessione in shell.

Ogni finestra avrà il proprio device associato (*vtty1, vtty2* ecc.), avrà il proprio tempo-macchina, il proprio «login» ecc. Va detto che all'interno di queste finestre ci si trova nelle stesse condizioni di chi si collega da remoto via seriale. La velocità è ridotta a 1200 baud (su schermo!), non si possono eseguire programmi GEM ecc. ecc.

Un quinto terminale «window» è il *vtty0*: deve essere caricato come *accessorio* all'accensione del computer, e permette di aprire una finestra shell all'interno di qualunque applicazione GEM.

Lo so che a parole le cose sembrano intricate, ma con il mouse in mano, sotto GEM il pull-down menu parlano da soli. Lo dico per quelli che in questo momento stanno pensando: «...ma questo di che diavolo parla???».

La C-Shell

Il command interpreter con cui si lavora sotto MTC è una C-Shell quasi del tutto compatibile con quella sotto *BSD Unix 4.3*.

Sarebbe troppo lungo descrivere da zero cos'è e come funziona una shell di questo tipo: mi limiterò ad illustrarne le principali caratteristiche.

Ha l'*history* (definibile), macrogenerazioni, variabili di *env* e normali, *alias substitution, filename, substitution* (con le wildcards), *command substitution*, redirezione di *Input e Output, path hashing, pipes*, nonché un utilissimo *file and command completion*. Quest'ultimo, non disponibile sotto Unix Sys V, è un «optional» assolutamente comodo: ogni qual volta dovete battere il nome di un file presente su disco, è sufficiente scrivere le iniziali e battere < ESC >. La shell capirà a cosa vi stavate riferendo. Se ci sono più file con le stesse iniziali provvederà ad indicarveli.

La C-Shell riconosce i file eseguibili da quelli leggibili soltanto dal SUFFISSO. Per la MTC tutti i file che terminano con un "*.app .prg .tos .ttp .sh*" sono potenzialmente «runnable». Dipende poi dalle condizioni di lavoro il poter lanciare o meno un'applicazione. Esempio: se ci si trova sotto Visual shell o da remoto (porta modem) e si prova a lanciare un programma che usa il GEM, la C-Shell risponderà «*you cannot run gem programs*» e non andrà oltre. Avrete notato il suffisso "*.sh*": con questo la shell riconosce i suoi scripts eseguibili. È chiaramente possibile scrivere dei programmi in shell, usando i comandi unix, e poi eseguirli. Il comando "*source*" forza l'esecuzione di un file come shell script, mentre se il file ha un nome che termina appunto con .sh, la shell lo riconoscerà subito senza bisogno di ulteriori specifiche.

Il listato di figura 1 è quello di un miniprogrammino che esegue il reset della macchina dopo «n» secondi, previo avviso a tutti gli utenti collegati.

Figura 1 -

```
%cat \etc\shutdown.sh
# Normal system shutdown
echo " " >\tmp\shut
echo "Warning: hardware shutdown in $1 seconds" >>\tmp\shut
echo "Close your files and logoff now!" >>\tmp\shut
\bin\wall <\tmp\shut
sleep $1
\bin\wall <\etc\shutmsg3
\usr\bin\reset
```

Il programma va lanciato da shell con il comando:

etc\shutdown 60 &

In tal modo dopo 60 secondi il sistema verrà resettato.

Gli utenti collegati riceveranno sullo schermo il messaggio:

Message from root on con: at 12..

Warning: hardware shutdown in 60 seconds
Close your files and logoff now!
... e pochi secondi prima del reset un ultimo avviso (il file «shutmsg3») segnalante l'imminente reset di tutto quanto.

L'intera procedura di installazione del sistema, contenuta sul disco originale, è scritta in shell, tanto per dare un'idea...

Quali sono i comandi disponibili da shell? Innanzitutto cominciamo con il precisare che la C-Shell, internamente, ha solo quelli riguardanti gli script e i processi. I comandi in realtà vengono per il resto caricati in memoria ed eseguiti proprio come normali programmi TOS (nonostante il suffisso .prg).

Ciò permette l'installazione di una MTC in formato ridotto anche su di un singolo floppy disk doppia faccia.

I comandi interni alla C-Shell sono:
alias, bg, break, cd, continue, echo, fg, foreach, end, goto, history, if, if then, else if then, endif, jobs, kill, logout, nice, printenv, pwd, set, setenv, source, stop, unalias, unset, while, end.

I comandi «di libreria», presenti nella directori *\bin* sono:

at	df	lpr	pick	stty
banner	diff	ls	pr	tail
cal	entab	mail	ps	tee
cat	fcp	man	rm	uniq
ccompil	fgrep	me	rmdir	wall
chmod	file	mesg	rpl	wc
cmp	find	mkdir	sed	who
cp	finger	more	show	write
csh	fmt	mv	sleep	
date	gem	news	sort	
dc	grep	od	split	
detab	head	passwd	strings	

Per ragioni di spazio non potremo esaminarli tutti.

***passwd** cambia la propria password, esattamente come su Unix.

***finger** riporta informazioni su di un utente specificato. Vedi esempio di figura 2.

***mail** permette di mandare a ricevere «posta elettronica» nel sistema. Ogni qual volta si entra nella MTC e ci sono nuovi messaggi in arrivo, la shell segnala «*you have mail*».

***man** permette di leggere i manuali in linea sul comando o sull'argomento specificato. I files di documentazione si trovano nella directory *\usr\man*. La MTC ha i manuali «elettronici» di TUTTI i comandi disponibili.

Ad esempio il comando *man tee* sarà equivalente a scrivere
more \ usr\ man\ tee.man

***me** è *microemacs*, l'*editor* «ufficiale» della MTC. Si tratta di un software ormai collaudato in un po' tutti gli ambienti: sotto VAX/VMS, sotto Unix, su sistemi operativi minori come l'OS/9 ecc. Microemacs permette lo «split» della finestra video in altre sottofinestre; permette la coesistenza di più testi (buffers), ha un vasto repertorio di comandi interni. Nel complesso è più potente di *vi*, nonché è meno complicato...

Figura 2

```
%finger dev
     Login name: dev              In real life: development team
     Directory: \usr\dev
     Shell: \bin\csh
     Plan:
     Programmazione in C
     Project:
     MT C Shell custom software
```

Microemacs si può utilizzare anche da remoto (collegati via modem alla MTC), purché si disponga di *emulazione vt100 (ansi)*.

***who** segnala quali utenti stanno utilizzando la MT C-Shell, e su quali terminali. Si ottiene un output di questo tipo:

```
%who
root     vtty1:   Thu Mar 24 20:07 1988
dev      vtty2:   Thu Mar 24 20:08 1988
demo      tta1:   Thu Mar 24 19:57 1988
%
```

***write** permette di entrare in comunicazione diretta con un altro utente collegato, il quale riceverà
"Message from .. on .. at.. ecc. ecc."

***mesg** abilita o meno la ricezione di messaggi sul proprio terminale (mesg y opp. mesg n).

***wall** non è altro che un «write» esteso a tutti gli utenti collegati.

Ci sono infine altri due comandi disponibili solo al «root» nella directory "\etc": *mkuser* e *rmuser*. Aggiungono o tolgono un utente dalla lista *\etc-passwd*.

File system e devices

Purtroppo il TOS non può assegnare un «proprietario» ad un file: non ha senso, capirete bene, su una macchina monoutente.

Ma sotto MTC le cose cambiano: due utenti ci stanno comodi, e con opportune espansioni hardware (aggiunta di seriali), si può arrivare a collegare fino a 5 terminali remoti. In questa logica il problema della protezione dei dati si fa vivo: se tutti possono leggere e cancellare i file di tutti, immaginatevi che scherzi...

La MTC non pone rimedio a questo «inconveniente»: l'unica soluzione consiste nel «nascondere» dalla vista degli altri i file privati, utilizzando l'opzione «hidden» o «system» (per il superuser) con il comando «*chmod*». È possibile rendere un file non-scrivibile (e quindi non-cancellabile), ma chiunque può rimettere le cose a posto, e combinare guai.

Poiché inoltre chiunque può eseguire un programma — posto che ne conosca l'ubicazione — anche da remoto (format, ship, reset ecc.) vi lascio immaginare che razza di sicurezza abbia questa shell.

Multiuser sì, senz'altro, ma «solo tra amici»!

Tornando al nostro file system, come abbiamo visto la struttura della MTC è solo apparentemente simile a quella Unix: «superblocco», «i-list», «i-nodes» non hanno molto senso in quest'ambito. A maggior ragione, i «files speciali», con il loro file system, non esistono del tutto. Anche i device vengono trattati in forma «custom».

I device della MTC sono:

con:	console
lpr:	stampante
tta1:	rs232 standard
tta2,3,4,5:	rs232 aggiuntive (multiserial card)
vtty0:	visual shell, accessorio
vtty1:	visual shell, default
vtty2:	
vtty3:	visual shells multiple
vtty4:	

Su ognuno di questi device è possibile redirigere sia Input che Output, esattamente come su Unix. Si sente tuttavia la mancanza di un */dev/tt..*: tutte le operazioni vengono infatti intercettate dalla C-Shell, che le distingue per i ":" finali.

Un programma non può contare sui corrispondenti «file speciali». Manca totalmente il */dev/null*. Anzi, la directory */dev* non c'è proprio!

Ho preferito mettere le mani avanti nell'illustrare l'uso della MTC, per non illudere inutilmente chi pensa di acquistarla: è solo «simile» a Unix, ma di mezzo c'è un abisso incolmabile. Questi erano i lati negativi della faccenda (alcuni, non tutti), passiamo adesso a quelli positivi (ce ne sono, per fortuna!).

Il multitasking

Il tanto sospirato ambiente multitasking *funziona*. Prima di tutto, però, occorre fare una considerazione: a cosa serve poter eseguire più programmi contemporaneamente? Dovete porvi questa domanda per poter cercare un valore effettivo della MT C-Shell. Dico subito che non è possibile eseguire applicazioni GEM contemporanee: solo ed unicamente TOS, e comunque *non interattive*.

Per inciso, non potrete far girare due programmi GEM in due finestre differenti, come qualcuno forse si aspettava.

Potrete invece lanciare applicazioni *non interattive* in totale background, senza rallentare di molto le prestazioni della macchina. Cosa intendo per «non interattive»? Qualunque .TOS che si comporta come .TTP e TUTTI i .TTP (TTP=take tos parameters).

Qualche esempio: *ARC.TTP, WXYMODEM.TTP, ZMODEM.TOS, KERMIT.TOS, cp.prg* (comando shell che esegue la copia dei file), ecc. ecc. ecc. fino ai normali *shell scripts*.

Il comando *nice* aumenta o diminuisce la priorità di esecuzione dei processi in corso e di quelli in background; si può specificare fino a 255 priorità differenti.

È da tenere presente che alcuni programmi GEM che non rispettano le chiamate al BIOS interferiscono con la MTC «congelando» a tratti gli altri processi in corso. Si tratta per fortuna di casi abbastanza rari.

Un uso molto comodo del multitasking — nel mio caso — è quello per trasferire file da/a computer remoti. In XMODEM, YMODEM, KERMIT o ZMODEM si può comandare il download, ad esempio, in background mentre contemporaneamente si edita un file o si gioca ad Asteroids!

Si può inoltre *compilare* in background, non uno, ma anche due, tre sorgenti, e contemporaneamente editarne un quarto!

Beh, si tratta di casi-limite, ma la cosa comunque funziona.

È necessario assegnare, alla propria shell la massima priorità di esecuzione con un bel «*nice-255*», ed eseguire gli altri processi con nice positivi.

Ad esempio per comprimere tutti i file della directory «\work» si può usare un comando tipo

```
nice +255 arc a paolo. arc "\work\*.*" &
```

La «*&*» sta per «esegui in background» appunto.

Volendo si può redirigere i messaggi su un file temporaneo da cancellare successivamente (tutti i file in *\tmp* vengono automaticamente rimossi dalla MTC al bootstrap).

Ad esempio per compilare il file "Shello.c" senza avere messaggi sullo standard output (il video), li destiniamo ad un file chiamato «*null*» nella directory *tmp*.

```
cc hello.c>\tmp\null &
```

Appena lanciato il processo, la shell ci segnalerà qualcosa tipo: *[1]9*.

È il numero di processo a cui dovremo riferire per compiere operazioni quali: l'interruzione, il «congelamento», il passaggio in foreground oppure per semplici controlli di stato.

Il comando *jobs* ci indicherà quali processi abbiamo lanciato, e se tenteremo di abbandonare la Shell con un *logout* il sistema ci segnalerà «*You have jobs running*» respingendo la nostra richiesta. Questo per impedire danni irrecuperabili: se il programma in back-

Un esempio di come la C-Shell presenta le directory

Notate che i «bit» di protezione in lettura scrittura, esecuzione sono a tre a tre uguali: il file system non supporta la struttura di Unix, anche se cerca di assomigliargli.

```
8 files (197336 bytes, 192 K), 7 directories
d--x--x--x        0  Feb 16 17:55 1988  bin
d--x--x--x        0  Feb 16 17:58 1988  etc
d--x--x--x        0  Mar 06 14:09 1988  lib
d--x--x--x        0  Mar 08 20:19 1988  mwc
d--x--x--x        0  Aug 13 19:00 1987  src
d--x--x--x        0  Feb 16 18:08 1988  tmp
d--x--x--x        0  Feb 16 17:58 1988  usr
-rw-rw-rw-      784  Mar 20 18:31 1988  lan.g
-rwxrwxrwx    88181  Mar 08 19:21 1988  lan.prg
-r-xr-xr-x    22611  Feb 16 18:01 1988  initl.prg
-rwxrwxrwx    25341  Feb 16 18:01 1988  init2.prg
-r-xr-xr-x    27875  Feb 16 18:01 1988  initvsh.prg
-rw-rw-rw-      328  Feb 21 04:16 1988  vsh.inf
-r--r--r--     1178  Feb 16 18:01 1988  vsh.rsc
-r-xr-xr-x    31038  Feb 16 18:01 1988  vshrun.prg
```

Situazione dei processi in tre differenti casi di boot:

In questo caso la lista è stata ottenuta MENTRE era in esecuzione il programma "FLASH.PRG" (per le comunicazioni).

- Single user -

```
PID PPID PRI STAT TTY     TIME COMMAND
  0    0  10 W           0:01 init
  1    0 110 W <   con:   0:20 \bin\csh
  2    1 110 R <   con:   0:00 ps
```

- Multiuser monoutente -

```
PID PPID PRI STAT TTY     TIME COMMAND
  0    0  10 W           0:01 init
  1    0   4 S N         0:00 cron
  2    0   4 W N         0:00 lpd
  3    0  10 W     con:  0:02 getty
  4    3  10 W     con:  0:20 \bin\csh
  5    4  10 R     con:  0:00 ps
```

- Visual shell con una sola finestra -

```
PID PPID PRI STAT TTY     TIME COMMAND
  0    0   1 W N         0:02 init
  1    0   4 S N         0:00 cron
  2    0   4 W N         0:00 lpd
  3    0   1 R N   con:   0:38 vsh
  4    3  10 W     con:   0:03 -getty.prg
  5    4  10 W    vtty1:  0:14 \bin\csh
  6    5  10 R    vtty1:  0:00 ps
```

- Visual shell come accessorio -

```
PID PPID PRI STAT TTY     TIME COMMAND
  0    0   1 W N         0:04 init
  1    0   4 R N         0:00 cron
  2    0   4 W N         0:00 lpd
  3    0  10 W     con:   0:54 vsh
  4    3  10 R     con:   0:00 -getty.prg
  5    3  10 W     con:   0:04 -getty.prg
  6    5 210 W <  vtty0:  0:24 \bin\csh
  7    3  10 R     con:   1:34 -FLASH.PRG
  8    6 210 R <  vtty0:  0:03 ps
```

Figura 3

```
%ps -lax
  PID PPID PRI STAT TTY     TIME COMMAND
    0    0  10 W           0:01 init
    1    0 110 W <    con:  0:29 \bin\csh
    2    1   1 W N         0:00 -csh
    3    2   1 R N         0:48 arc
    4    1   1 W N         0:01 -csh
    5    4   1 R N         0:06 arc
    6    1 110 R <    con:  0:00 ps
%jobs
   [1] + Running    nice +255 arc awn balance.arc *.* >e:\tmp\null
   [2] - Running    nice +255 arc xwn starnet >e:\tmp\nul2
```

*La colonna «PRI» indica le priorità di esecuzione: notate il PID 1 (csh del sottoscritto), con un bel 110 di PRI, notate il PID 6 (il comando **ps** appena eseguito), lanciato con la stessa priorità del padre (PPID) 1. Notate infine i PRI dei processi 2, 3, 4, 5: sono a 1. Infatti mediante «**nice**» avevo volutamente rallentato la loro velocità, a vantaggio della MIA, per non subire rallentamenti di sorta.*

ground stava per esempio scrivendo su disco, una brusca interruzione avrebbe lasciato la scrittura a metà, magari rendendo il disco illeggibile.

È possibile controllare lo stato dei processi con il comando Unix «*ps*»: nell'esempio di figura 3 riporto quello che la MTC rispondeva durante 2 (due) archiviazioni-compressioni su hard disk lanciate in background: la prima comprimeva tutti i file della directory corrente in uno unico chiamato «balance.arc», sopprimendo messaggi di errore, e comunque mandando l'output su un file temporaneo chiamato «null» in \tmp; la seconda estraeva dall'archivio «starnet.arc», i file contenuti e li metteva nella directory corrente.

I comandi per la gestione dei processi sono in tutto 7:
ps, bg, fg, jobs, stop, kill, nice
— **stop** {numero_del_processo} «congela» in memoria il processo indicato. L'esecuzione è solo interrotta momentaneamente, e può essere ripresa (utile quando si ha bisogno di velocità e c'è un processo molto lungo in corso).
— **kill** {numero_del_processo} toglie il task indicato dalla lista di esecuzione. Il processo «muore» irrevocabilmente. In teoria la memoria dovrebbe venire rilasciata, in pratica purtroppo non sempre succede.
— **bg** rimette in esecuzione un processo fermato con *stop*, sempre mantenendo il background.
— **fg** rimette in esecuzione un processo fermato con *stop*, portandolo però in *foreground*.
— **nice** assegna la priorità di esecuzione di un processo.
— **ps** e **jobs** listano i processi generali e quelli solo in background.

È evidente che ogni utente (posto che ce ne sia più di uno) può solo manipolare i propri processi. Il comando «*nice*» inoltre non può essere impartito con valori negativi (che per la MTC significano una maggiore priorità di esecuzione) se non dal super-user, o da chi ne ha la password (*su root*).

Il *root* può «*kill*are» qualunque processo, senza distinzioni, anche quelli critici come i getty e l'init. In genere l'uso sconsiderato del *kill* finisce col provocare un grandioso «shutdown» con ricchi messaggi di disperazione da parte del Kernel, che non sa più che pesci pigliare.

La *velocità di esecuzione* dei nostri processi è *realmente controllabile*, e darà luogo a gradite sorprese.

Distinguiamo innanzitutto i processi «pesanti» da quelli «leggeri»: i primi, quali l'*ARC*, sovraccaricano di lavoro la CPU: i secondi, come i WP e tutti i text editor in genere, rimangono quasi sempre in attesa in un INPUT. Con il comando *ps* si può notare l'effettivo utilizzo in secondi della CPU: la *csh* dell'esempio precedente ha «consumato» in tutto 29 secondi-CPU, nonostante sia in uso da qualche ora. Il processo di ARC, in esecuzione da pochi minuti, ha già «consumato» 48 secondi-CPU. Eppure mentre sto editando un testo, non noto ALCUN rallentamento: non appena faccio qualcosa tutto il mio tempo-macchina mi viene restituito secondo le priorità definite.

La lista di processi pubblicata in figura 4 è quella relativa ad una sessione di visual shell (due per l'esattezza): sul terminale virtuale *vtty1* e sul *vtty2* ho lanciato il benchmark *Dhrystones* a distanza di pochi secondi l'uno dall'altro, ma con priorità di esecuzione differenti (entrambi in background). In pratica nel Desktop comparivano due finestre semi-sovrapposte che riportavano fedelmente due sessioni differenti. Con il mouse ho «clickato» sulla prima, facendola diventare «attiva» (prende comandi dalla console), e ho lanciato il "dhryst.prg" (processo 10, terminale vitty1). Poi ho «clickato» sulla seconda finestra, e su quel terminale (*vtty2*) ho fatto la stessa cosa, con un «nice» minore (processo 12).

Il processo 0 è l'*init*, padre di tutti i processi. Il proc.1 è il *cron*, il 2 e l'*lpd* (il printer spooler), il 3 è il *visual shell manager*, il software di interfacciamento MTC-GEM.

Il 4,6,8 sono le *getty*: perché tre? Semplice, avevo aperto una terza sessione sul terminale *vtty3*, ma per problemi di memoria ho dovuto interromperla con un *logout*. Sulla finestra è ricomparso il classico «login:», che permane anche se fisicamente il terminale è «spento». Il getty continua a funzionare su quel terminale (e occupa memoria..).

Abbiamo poi le C-Shell e i due «dhryst» in funzione.

Al termine della prova, il primo mi ha dato come risultato 350 Dhrystones, il secondo 270. Considerato il fatto che erano in background, con priorità ridotta, su due terminali differenti, SOTTO GEM, non ci si può proprio lamentare

Figura 4

```
%ps -lax
  PID PPID PRI STAT TTY     TIME COMMAND
    0    0   1 W N         0:04 init
    1    0   4 S N         0:02 cron
    2    0   4 W N         0:00 lpd
    3    0   1 R N    con: 7:06 vsh
    4    3  10 W     con:  0:04 -getty.prg
    5    4 125 W < vtty1:  0:35 \bin\csh
    6    3  10 W     con:  0:04 -getty.prg
    7    6  10 R   vtty2:  0:22 \bin\csh
    8    3  10 R     con:  0:06 -getty.prg
    9    5   1 W N         0:01 -csh
   10    9   6 R N         0:45 dhryst
   11    7   1 W N         0:01 -csh
   12   11   1 R N         0:30 dhryst
   13    5 125 R < vtty1:  0:03 ps
```

della velocità. Un IBM-XT mi risulta ottenga dei valori attorno ai 400. Il Dhrystone eseguito in foreground su MTC con massima priorità ottiene da un minimo di 1070 ad un massimo di 1250 (a seconda del compilatore).

Quanti e quali programmi possono «girare» in background?

Il numero è vincolato solo dalla memoria disponibile (si deve tenere presente ancora una volta che il sistema operativo si riserva dello spazio in più per ogni processo attivo): con una opportuna amministrazione dei tempi di esecuzione, la CPU non sarà mai eccessivamente lenta nei confronti del proprio processo corrente.

Per i «quali» non c'è una risposta precisa: dipende. Nonostante la Beckemeyer garantisca il funzionamento in background di soli programmi TOS (senza GEM), io ho lanciato con successo un demo grafico tridimensionale in background, ottenendo un effetto curioso: avevo sì il prompt della C-Shell, ma sullo schermo si «agitano» linee e punti che andavano a disegnare le immagini originali. Quindi funzionava normalmente!

Alcuni programmi che necessitano di un input limitato (numeri o lettere, in genere), possono essere lanciati non interattivamente fornendo loro i dati con una redirezione di input appunto.

Ad esempio, se il programma «conta-.tos» mi chiede all'inizio di inserire il numero di ripetizioni di un ciclo, posso fornire tale numero al processo con il comando.

%conta.tos <miofile &

dove «miofile» contiene il numero dei cicli che ho deciso di fare eseguire al programma «conta». *Tutto questo rientra nella normalità per chi è abituato a lavorare sotto Unix.*

Anche i *compilatori C «Megamax»* e *«Mark William's»* possono essere lanciati in background senza alcun problema.

Attualmente io uso il Mark William's (per il mondo IBM produce un compilatore chiamato «*Let's C*»), che addirittura cita nel manuale l'uso della MTC alternativo a quello della sua mini-shell. Poiché questo compilatore supporta anche chiamate «system», lettura dell'environment ecc. ecc. ne risulta che si ha veramente l'impressione di lavorare sotto Unix.

Alcuni software per la gestione di banche dati e BBS (*FoReM* o *STadel*) sono stati scritti per poter essere eseguiti in background sotto MTC.

Non dimentichiamoci infine gli «shell scripts»: anch'essi girano in background senza intoppi. Purtroppo si «trascinano» dietro altri 66K di C-Shell, che li esegue materialmente, per cui non sono estremamente convenienti su un sistema da 1Mega come il 1040..

L'*accesso ai dischi* è controllabile solo parzialmente: la *lettura* può essere fatta normalmente anche in background senza che la shell si blocchi. Si può cioè comandare il *load* di un intero disco su hard disk in *background*. Purtroppo il *format* e tutte le scritture in genere "congelano" i processi fino al termine del lavoro. La prima cosa che ho provato appena ricevuto il package è stata proprio la formattazione di un dischetto («..finalmente posso formattare senza perdere 2 minuti ogni volta..»), ma senza successo: mi si è inchiodato tutto..

Questo vale ovviamente *per i disk drive*: con l'hard disk non si hanno problemi di velocità, di solito.

La condivisione dei file funziona a meraviglia: è possibile far leggere contemporaneamente a più processi lo stesso file (ovvio), oppure fare che uno legge mentre l'altro scrive, oppure più processi che scrivono sullo stesso file (ne viene creata una copia identica e ogni processo lavora su quella, datata diversamente per il riconoscimento).

Un discorso particolare meritano il «*cron*» e l'*at*. Il «cron» esegue dei programmi specificati con una frequenza definita: è possibile ad esempio programmare la MTC perché ogni 10 minuti salvi il contenuto di una certa directory dell'hard disk su floppy (solo un esempio). L'«at» esegue un programma (o un comando, che è poi la stessa cosa) ad una certa ora di un certo giorno definito; si può specificare anche una periodicità settimanale, mensile, continua, ecc. ecc. Ad esempio si può comandare uno «shutdown» ad una certa ora, oppure (utile) si può decidere di attivare il modem in autoanswer sulla porta seriale ad un'ora, e disattivarlo ad un'altra.

La gestione della memoria

Eccoci alle dolenti note. Sembra impossibile, eppure con 1 Mega a disposizione si combina veramente poco! Praticamente preclude l'utilizzo della Visual Shell, e lascia poco spazio per un utiliz-

Il file "\etc\passwd"

I campi sono separati da ":": il primo indica lo user name, il secondo rappresenta la password (encryptata), il terzo ed il quarto rispettivamente user-id e group-id, su Unix fondamentali per l'accesso ai file, mentre su MTC solo apparenti. Il quinto campo contiene il nome reale dell'utilizzatore, il sesto la home directory, il settimo il programma da eseguire al login (solitamente l'interprete di comandi **csh**).

```
%cat \etc\passwd
root::0:0:The Superuser:\usr\root:\bin\csh:524287:5
trouble:hSPABAXA:1:0:Superuser trouble entry:\:\bin\csh:524287:5
who::2:0:who is on line:\etc:\bin\who:524287:5
ps::3:0:whats going on:\etc:\bin\ps:524287:5
bin:NOLOGIN:4:0:system owner:\bin:\bin\csh:524287:5
admin:NOLOGIN:5:0:administrator:\etc:\bin\csh:524287:5
com::6:0:communications:\usr\com:\usr\bin\com:524287:5
guest:HcFAtqKA:100:100:Guests account:\usr\guest:\bin\csh:524287:5
silos:wggApCJA:102:100:Paolo Pennisi:\usr\silos:\bin\csh:524287:5
eddy:cZuAdJXA:100:200:X25 OS9 Project:\usr\x25:\bin\csh:524287:5
steve:tQNABXDA:105:100:Stefano Borelli:\usr\x25:\bin\csh:524287:5
dev::11:0:development team:\usr\dev:\bin\csh:524287:5
```

zo biutente. Il Kernel supervisore (*RXTBOOT.PRG*) è lungo 34K, e viene lanciato al boot del MTC. In single user viene caricata la C-Shell (66K), e naturalmente INIT1 (22K). In tutto 122K: ma si deve ben considerare che TUTTI i comandi che useremo dovranno essere CARICATI da disco in memoria per essere eseguiti, e generalmente la dimensione media di un comando si aggira attorno ai 20K. Direi che in single user mode 150K costituiscano l'effettiva occupazione di memoria del sistema operativo.

In *multiuser mode (monoutente*, cioè con la *tta1*: *disabled* dobbiamo aggiungere il *getty* (19K), l'*init2* (25K), *atrun* (12K), i daemon del *cron* e *lpr*: circa 200K effettivi durante il funzionamento.

In *multiuser biutente*, con il secondo terminale sulla porta *tta1*: (la RS232, lo ricordo), dobbiamo aggiungere ancora: un *getty* ed una *C-Shell*: circa 100K effettivi alla fin fine da aggiungere ai 200 di prima, in totale *300K per 2 utenti. Per ogni utente aggiuntivo occorrono circa 100K.*

Insomma, di quel Mega con cui eravamo partiti, rimangono circa 400-450K, da dividere fraternamente in due (sto considerando l'uso «*multiuser*» con un ipotetico secondo terminale remoto).

```
TEE(1)                  MICRO C-Shell Manual                  TEE(1)


NAME
     tee - provide a T junction in a pipeline

SYNOPSIS
     tee file

DESCRIPTION
     Tee copies the standard input to the named file and to the
     standard output.  Thus

          ls -l | tee filelist

     shows a full listing of the contents of the directory on the
     console, and also puts it in the file 'filelist'.

CAVEATS
     Obviously, this command would be useful with any process whose
     output goes to the screen (standard output), but may cause
     unforseen side effects if used with commands that do not write
     to standard output.  Use it advisedly.
```

Bene, i 200K a testa sono insufficienti per la maggior parte delle applicazioni, che oltre ad occupare memoria fisicamente, ne richiedono altra per i dati da gestire.

Utilizzando la MTC come unico utente, invece, quei 450K non daranno problemi.

A quanti si staranno in questo momento domandando «ma non può fare lo *swap* su disco?» rispondo: no, non può perché il TOS stesso non può farlo. La MTC *non è un sistema operativo*, ma *solo un Kernel* che supervisiona il vecchio *TOS*.

Va detto che, durante il suo uso, la MTC continua ad allocare e poi rilasciare aree di memoria in cui carica ed esegue, ad esempio, i comandi che battiamo. In alcune situazioni, piuttosto frequenti in verità, la memoria rilasciata *non viene più riconosciuta* dal sistema, che in breve rimane senza!

Di punto in bianco, battendo *ls*, la MTC risponderà «*not enough memory to load \ bin\ ls.prg*». Soluzioni? Solo una: uscire dando un shutdown e rilanciare la MTC. Il GEM quando riprende il comando delle cose evidentemente ripristina i puntatori ai segmenti di memoria. Beh, meglio di niente, comunque è un problema tutt'altro che trascurabile!

E quando usiamo le *visual shells*? Eh, beh, velo lascio immaginare! Le *VSH* occupano in più *31K*, più altri *20K* di «lavoro» per le finestre. Si rimane a secco prima di accorgersi quello che succede. L'uso dell'accessorio «*multiwindow*», poi, è pura utopia.

Morale: *ci vogliono 2mega!*

Per sfruttare a pieno le capacità della MTC ci vuole MEMORIA, più ce n'è meglio è (bella forza). Dopo le frustrazioni dei «not enough memory» sul mio 1040ST, ho provato ad installare la MTC su di un Mega-4ST. Beh, non credevo ai miei occhi: ho aperto 4 sessioni in «visual», su ognuna ho lanciato in background un processo a bassa priorità, poi ho aperto la «multiview» (accessorio), e sono entrato in shell con un quinto terminale virtuale. La lista dei processi in corso era lunghina, ma tutto funzionava a meraviglia! Dal Desktop ho lanciato *Wordplus* (word processor piuttosto voluminoso), e ho caricato 2 testi.

Ho diminuito la dimensione delle due finestre GEM, e ho aperto dagli accessori la «multiview»: *dentro* Wordplus mi si è aperta la finestra con la Shell di prima (*vtty0*), e ho constatato che gli altri terminali stavano effettivamente lavorando normalmente. Ho lanciato in background un «*arc*» di 100 e rotti file, ho chiuso la finestra Multiview e ho continuato con Wordplus.

L'Hard Disk era perennemente all'opera, ma quando da Wordplus richiedevo la lista dei file, il sistema divideva gli accessi della testina secondo la giusta misura.

Se pensate che nello stesso momento un *sesto terminale* avrebbe potuto essere attaccato alla porta seriale, e magari editare un file e compilarlo...

Considerazioni finali sulla MT C-Shell

Nonostante forse ne sentiate parlare per la prima volta, in realtà la Beckemeyer C-Shell è disponibile già da MOLTO tempo. Negli USA è talmente conosciuta che gli stessi manuali di molti compilatori C (come il Mark William's) e di moltissimo software applicativo la citano come esempio di compatibilità.

Potremo anzi dire che è diffusa e conosciuta quanto nel mondo IBM lo è il *Concurrent DOS* o *Multilink*.

Chi cerca il multitasking ma non vuole perdere la biblioteca di programmi GEM e TOS che ha disposizione, trova nella MTC la soluzione di tutti i suoi problemi.

La domanda che qualcuno si potrà porre è: «come mai in Italia non è mai stata presentata?».

Semplice, perché l'Atari non ha ritenuto «interessante» il prodotto. Fortunatamente in tutti gli altri paesi non la pensano così...

A chi può servire la MTC? Principalmente a tutti gli sviluppatori di software in C (che — anzi — non dovrebbero proprio farne a meno), per i quali esistono utility molto potenti come il *make*.

Ma anche a chi in generale sviluppa software che necessita di un editor e di un compilatore.

A chi avanza un terminale (magari il vecchio personal che sta ammuffendo

Contenuto del package

MT C-Shell	1 disco da 720K, manuale ca.	100 pagine
VSH manager:	1 disco da 360K, manuale ca.	30 pagine
C-tool:	1 disco da 360K, manuale ca.	10 pagine
manuals online:	1 disco da 360 K	

Note sulla documentazione

Il manuale della MT C-Shell è essenzialmente rivolto ad un utente già pratico di Unix. La descrizione dei singoli comandi è prettamente illustrativa. Non viene trattata la programmazione in shell. Il file **\etc\inittab** non viene citato, eppure è letto dal sistema al bootstrap multiuser regolarmente.

Il manuale — non essendo il programma distribuito nel nostro Paese — è ovviamente in inglese. È consigliata la lettura di un testo su Unix.

Prossime release

Annunciata la versione «network» che permetterà la condivisione delle periferiche tra macchine Atari. Upgrades del sistema operativo pagando il solo costo della spedizione.