Come funziona Linux

Terza parte

Processi, gestione della memoria e librerie

In questa puntata studieremo gli strumenti che Linux mette a disposizione per la gestione dei processi. Vedremo come il kernel gestisce la memoria ed i meccanismi di ottimizzazione nell'uso della stessa. SI tratta di un argomento abbastanza teorico, ma se avrete la pazienza di leggere fino in fondo imparerete molte cose interessanti.

di Giuseppe Zanetti

Programmi e processi

Col termine processo si intende una singola istanza di un programma in esecuzione nel sistema. Essendo Linux un sistema multitasking, è possibile che in un dato momento stiano funzionando contemporaneamente più copie dello stesso programma. La concorrenza fra processi (ovvero il fatto che i processi girino contemporaneamente) implica, a livello di kernel, tutta una serie di accorgimenti atti ad evitare conflitti durante la condivisione della memoria o delle diverse periferiche hardware.

Il semplice fatto di accedere ad una macchina con la procedura di login, causa l'esecuzione contemporanea di più processi da parte dell'utente, che si andranno ad aggiungere a quelli, di cui non scorgiamo l'esistenza, che vengono lanciati al boot della macchina da init e rimangono attivi in background (sottofondo).

Ogni processo può mandare in esecuzione più processi figli. Ad eccezione di init, ogni processo ha a sua volta un unico processo padre. Init è particolare, essendo il progenitore (non il "padre") di tutti gli altri processi. Il kernel assegna ad ogni processo un numero progressivo, detto PID (Process IDentifier, identificatore di processo), con lo scopo di poterlo poi identificare univocamente.

Processi in foreground e background

Esistono due modalità di esecuzione di un processo: la prima, detta foreground (in primo piano), consiste nel legare il processo ad una tty (terminale), che può avere un corrispondente fisico, ad esempio la console della macchina, oppure essere una porta virtuale, ad esempio quando si accede alla macchina usando telnet. Sul terminale associato al processo vengono diretti l'output e i messaggi di errore generati dal programma e mediante esso l'utente può fornire dati o controllare il processo.

È poi possibile eseguire un processo in background (sottofondo), in modo che esso continui il proprio lavoro senza tenere impegnata una porta. In questo caso generalmente l'input e l'output prodotti vengono ridirezionati su file. È questo il metodo di funzionamento ad esempio di tutti i programmi che offrono servizi di rete (web server, ftp server, ...) o che sovrintendono a funzioni del sistema (schedulatore di attività, spooler di stampa, ...).

In una sessione di lavoro da shell è possibile eseguire un programma in foreground semplicemente digitandone il nome e i parametri necessari al suo funzionamento sulla linea di comando:

```
$ ls -l
```

in questo caso il prompt di richiesta di nuovi comandi tornerà all'utente solo al termine del processo.

Volendo invece far lavorare un processo in background è necessario aggiungere in coda alla linea di comando il carattere &. In questo caso è opportuno ridirezionare su file l'input e l'output del processo, mediante gli operatori < e >:

```
$ cruncher <input.txt >output.txt 2>errori.txt &
[1] 7126
$
```

Il messaggio [1] 7126 viene generato dalla shell e ci indica che al processo è stato assegnato il PID 7126. Si noti che così facendo il controllo viene restituito immediatamente all'utente, che può da subito inserire nuovi comandi.

Quando la shell termina, ad esempio perché l'utente ha impartito il comando exit, generalmente viene inviato un segnale di terminazione anche a tutti i processi in background ad essa associati. Per evitare questo inconveniente ed eseguire un comando in modo slegato dalla shell che l'ha lanciato, si può utilizzare il comando nohup:

```
$ nohup cruncher <input.txt >output.txt &
```

Se non altrimenti specificato, nohup ridireziona l'output del comando sul file nohup.out.

Comandi di gestione dei processi

Per accedere alle informazioni sui processi si utilizza il comando ps:

```
$ ps -axl
```

F	UID	PID	PPID	PRI	NI	SIZE	RSS	WCHAN	STAT	TTY	TIME	COMMAND
100	0	1	0	30	15	872	148	111aef	S N	con	0:15	init auto
40	0	2	1	30	15	0	0	12770f	SWN	con	0:00	(kflushd
40	0	3	1	42	27	0	0	1201c9	SWN	con	0:00	(kswapd
40	0	7	1	30	15	0	0	1760ee	SWN	con	0:00	(nfsiod
140	0	12	1	30	15	836	68	109f04	S N	con	0:01	update
140	0	34	1	30	15	876	196	109f04	S N	con	0:07	crond

Oltre al PID del processo, vengono listati il PID del processo padre (PPID), lo UID dell'utente proprietario, la priorità di funzionamento (PRI), la tty associata (TTY), il tempo macchina utilizzato (TIME), la linea di comando (COMMAND) ed altre informazioni utili.

Solamente l'utente proprietario e root possono uccidere o cambiare priorità ad un processo. L'informazione sull'utente proprietario è necessaria anche per la gestione dei permessi di accesso ai file.

Altri comandi utili per ottenere informazioni sui processi sono top, il quale, oltre a fornire in tempo reale molte informazioni utili sullo stato del sistema, permette di vedere i processi attivi e di interagire con essi, e pstree, che disegna un albero dei processi in esecuzione sul sistema.

Uccisione di un processo

Per terminare un processo si utilizza il comando kill, che invia ad esso un "segnale". Se non altrimenti specificato, viene inviato il segnale SIGTERM, con cui si invita il processo a terminare in modo pulito le proprie operazioni.

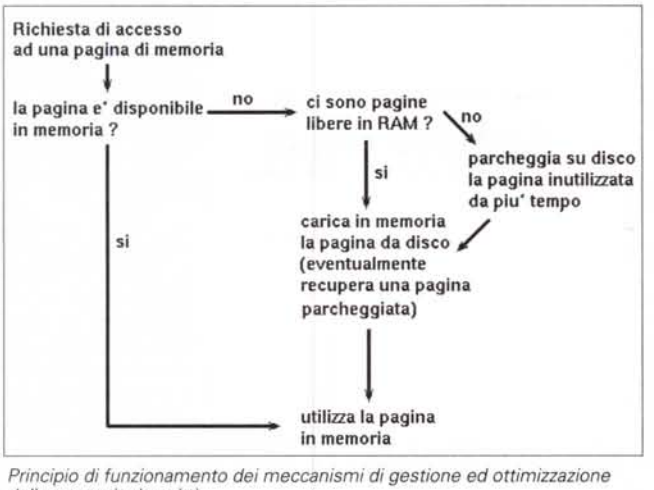
```
$ kill 1292
```

Il numero che segue il comando è il PID del processo, ricavato dal comando ps.

Se il programma non collabora, è possibile terminarlo forzatamente inviandogli il segnale SIGKILL:

```
$ kill -SIGKILL 1292
```

Questo metodo alle volte risulta tuttavia un po' troppo brutale, poiché non offre al programma l'opportunità di terminare in modo pulito il proprio lavoro.



Principio di funzionamento dei meccanismi di gestione ed ottimizzazione della memoria descritti.

Modifica della priorità di un processo

Il comando nice permette di associare ad un processo una priorità diversa da quella standard. I valori possibili vanno da -20 (massima priorità) a 19 (minima priorità).

```
$ nice -10 comando argomenti
```

Un utente normale può, ovviamente, solo lanciare processi con priorità minore o uguale a quella standard (corrispondenti a valori dell'indice più elevati).

Eseguire un processo con bassa priorità può essere utile per lasciarlo lavorare in background senza infastidire troppo il funzionamento del resto del sistema.

La gestione dei processi secondo POSIX

La shell di Linux, bash, offre anche la possibilità di gestire i processi (in questo contesto vengono chiamati "jobs") secondo lo standard POSIX Job Control.

Mentre un processo è in esecuzione in foreground è possibile sospenderlo e tornare al prompt della shell mediante la pressione contemporanea dei tasti Ctrl+Z. Una volta riottenuto il controllo della shell è possibile gestire lo stato del job sospeso utilizzando uno dei seguenti comandi:

```
$ jobs
```

Lista i job sospesi o in background.

```
$ bg [job]
```

Manda in background un job sospeso.

```
$ fg [job]
```

Permette di riprendere l'esecuzione in foreground di un job sospeso.

In tutti i casi, se non altrimenti specificato, l'operazione verrà effettuata

sull'ultimo job sospeso.

L'esempio che segue mostra come mandare in background una lunga sessione di ftp.

```
# ftp ftp.profuso.com
Connected to ftp.profuso.com.
220 ftp.profuso.com FTP server (Linux freddy 2.0.35 #5 Tue Nov 10 23:26:37 MET 1998 i586) ready.
Name (ftp.profuso.com:root): anonymous
331 Password required for anonymous.
230 User anonymous logged in.
331 Guest login ok, send e-mail address as password.
230-Next time please use your e-mail address as your password
230-        for example: joe@localhost
Password (ftp.profuso.com:anonymous):
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd pub/Linux
250 CWD command successful.
ftp> prompt n
Interactive mode off.
ftp> mget *
[1]+  Stopped                 ftp ftp.profuso.com
bash# bg
[1]+ ftp ftp.profuso.com &
bash#
```

La protezione della memoria

Una volta avviato, ogni processo funziona con i permessi dell'utente che l'ha lanciato e in modo separato dagli altri processi e dal kernel, in modo da evitare interferenze indesiderate (si pensi al caso di un programma difettoso che vada a scrivere nelle aree riservate ad altri). Per far ciò, Linux sfrutta i meccanismi di protezione della memoria messi a disposizione dalla MMU (Memory Management Unit) presente nell'hardware della macchina. Essendo tali meccanismi implementati in hardware, la protezione assicurata è completa. Alla partenza del processo, ad esso viene riservato un certo numero di pagine di memoria dove caricare il codice del programma e dove mantenere i propri dati. Una volta in esecuzione, se il processo tenterà di accedere ad aree di memoria al di fuori di quelle ad esso riservate, l'hardware della macchina segnalerà al sistema operativo una situazione d'errore e quest'ultimo la passerà al processo sotto forma di un segnale. A questo punto la routine di gestione del segnale si regolerà di conseguenza, ad esempio interrompendo il programma e restituendo all'utente una opportuna segnalazione di errore.

```
$ mioprogramma
Memory fault: core dumped
```

Generalmente la terminazione anomala di un programma è accompagnata dalla generazione di un file, di nome

Appunti Linux (autore: Daniele Giacomini)

di Giuseppe Zanetti

State cercando un libro su Linux che sia contemporaneamente in italiano, completamente libero, scaricabile gratuitamente dalla rete, costantemente aggiornato e, ovviamente, ben fatto? Allora smettete di cercare e andatevi a prelevare la nuova versione di AppuntiLinux, di Daniele Giacomini, un libro che non dovrebbe mancare nella biblioteca di qualunque utente serio di Linux.

Non fatevi ingannare dal nome: AppuntiLinux è nato sì come la raccolta degli appunti di un autodidatta, ma poi il buon Daniele non si è di certo sprecato e ben presto ha trasformato le poche pagine iniziali in un'opera grandiosa, che comprende praticamente tutto quello che un utente normale potrebbe desiderare conoscere su Linux e sulle applicazioni che vi girano.

Conscio della dimensione del suo lavoro e della quantità di pagine da scaricare e stampare, l'autore si è prodigato per rendere disponibile l'opera in diversi formati, utilizzabili a seconda delle necessità dell'utente: oltre alla versione HTML sono infatti disponibili i file in formato PDF, per la lettura a video, e Postscript, per la stampa. Quest'ultima a sua volta è disponibile anche in due formati adatti ad un utilizzo come strumento di tutti i giorni: è infatti possibile ottenere una impaginazione di due o quattro pagine su un singolo foglio A4. Non sarà troppo elegante ma è certamente molto funzionale. Chi non se la sentisse di stampare le 2000 e più pagine oppure avesse necessità di studiare solamente un singolo aspetto di Linux, può limitarsi a stampare i singoli tomi o addirittura i singoli capitoli o argomenti. Esistono infine alcune copisterie e tipografie che possono fornire ad un costo accettabile il volume già stampato. Per maggiori dettagli si consultino le pagine introduttive del libro.

La copertina del libro AppuntiLinux

L'aggiornamento dell'opera è costante e le nuove versioni vengono rese disponibili abbastanza frequentemente, per cui può valere la pena tornare spesso nel sito http://www.pluto.linux.it/ildp/AppuntiLinux per vedere se sono stati fatti nuovi aggiornamenti, in modo da tenere aggiornata questa documentazione con la versione di Linux che si sta utilizzando. La versione attuale del libro comprende già le funzioni disponibili nel kernel 2.2.

La spiegazione dei diversi argomenti vie-

"core", contenente un dump della memoria occupata dal programma. A posteriori sarà possibile utilizzare su questo file un debugger per scoprire le cause del malfunzionamento.

Comunicazione fra processi

La comunicazione e la sincronizzazione fra i processi non può essere effettuata direttamente, in quanto il meccanismo di avvio (fork) assegna ad ognuno delle proprie aree di memoria non accessibili direttamente ad altri.

Per ovviare a questo problema esiste la libreria standard IPC (Inter Process Communication), che fornisce diversi metodi di comunicazione fra processi: pagine di memoria condivisa, code di messaggi, segnali, mailbox e semafori. Altri metodi di comunicazione molto utilizzati sono i pipe e i socket.

I segnali

I segnali sono eventi asincroni che possono essere spediti ai processi. In concomitanza della ricezione di un segnale, il processo manderà in esecuzione una apposita routine di gestione. Un programma ben scritto prevederà ad esempio che in concomitanza della ricezione di un segnale SIGTERM venga eseguita una routine che si occupi di chiudere in modo pulito i file aperti. Se non viene definita una routine di gestione per un dato segnale, di solito si termina il programma oppure si ignora il segnale.

Da shell è possibile spedire un segnale ad un processo mediante il comando kill, che abbiamo già incontrato. Alcuni segnali vengono generati dal sistema in caso di eventi particolari, ad esempio in caso di tentativo di accesso ad un'area di memoria vietata (SIGSEGV) oppure di errore di calcolo da parte del coprocessore matematico (SIGFPE).

Alcuni segnali possono poi essere generati direttamente dalla shell che controlla il processo: ad esempio la pressione della sequenza di tasti CTRL+C manda al processo un segnale di SIGTERM.

Non necessariamente i segnali servono per terminare un processo: il segnale SIGHUP, ad esempio, viene spesso utilizzato per forzare la rilettura dei file di configurazione di un programma:

```
$ kill -SIGHUP named
```

I pipe

I pipe (tubi) consentono una comunicazione fra processi mediante un meccanismo di code. Il kernel si occupa di gestire la sincronizzazione nello scambio dei dati: se la coda è piena il processo che scrive rimane in attesa, viceversa se la coda è vuota rimane in attesa il processo che legge. I pipe possono essere gestiti passando attraverso un file speciale (named pipe), che permette la comunicazione fra processi qualunque, oppure mediante strutture interne al kernel (unnamed pipe) che consentono solamente la comunicazione fra processi strettamente correlati (padre e figlio).

Per creare un named pipe si utilizza il comando mknod:

```
$ mknod /tmp/miopipe p
```

Proviamo ora a creare un processo che mediante il comando cat legga i dati provenienti dal pipe:

```
$ cat /tmp/miopipe
```

In un altro terminale o console virtuale (ALT+Fn) proviamo ora a lanciare un qualunque comando ridirezionandone l'output in modo che vada a scrivere nel pipe:

ne fatta in modo completo e professionale, con uno stile estremamente curato e senza lasciarsi andare a traduzioni fantasiose dei termini inglesi. Una sezione del libro è interamente dedicata all'arte di scrivere e tradurre della documentazione tecnica.

A prima vista potrebbe sembrare un inutile spreco di tempo e di carta, ma si deve tenere in considerazione che il testo di Giacomini è nato nell'ambito del progetto ILDP (Italian Linux Documentation Project), nella cui mailing list l'argomento "stile di traduzione" è stato e viene tuttora trattato ampiamente e con una certa serietà.

Gli argomenti trattati su AppuntiLinux sono i seguenti:

- introduzione all'uso di Linux
- installazione di Linux (Red Hat, Slackware, Debian)
- architettura del sistema (kernel, filesystem, shell, terminali, utenti, ...)
- utilizzo del sistema (file, programmi, stampa, ...)
- grafica
- servizi di rete e internetworking
- modem, porte seriali, PPP
- scrivere con Linux (editoria elettronica, controllo ortografico, stile)
- programmazione (algoritmi, linguaggi più comunemente usati in Linux)
- linguaggi di programmazione specifici (espressioni regolari, awk, sed, M4, SQL)
- servizi di rete in dettaglio
- e-mail e Usenet
- sicurezza nella rete.

Nel capitolo riguardante l'editoria, una parte significativa è dedicata all'utilizzo del tool SGML e dello stile di impaginazione Doc Book, utilizzati per creare AppuntiLinux ed altra documentazione del progetto Linux Documentation Project.

Sotto l'aspetto didattico, l'opera è sicuramente valida: gli argomenti sono trattati in modo scorrevole ma tuttavia completo e, dove lo spazio non basta, viene sempre fornita una completa bibliografia di testi dove approfondire i concetti. Tali segnalazioni vengono fatte argomento per argomento al termine della spiegazione e ciò facilita enormemente il lavoro di ricerca. Ove vengano presentati degli esempi tratti da altri autori, Daniele è sempre ligio nel segnalare la fonte.

Il libro, che viene distribuito secondo la licenza GPL, può essere tranquillamente distribuito e copiato, ad esempio per utilizzarlo in corsi e attività di formazione. Il libro di Giacomini è un ottimo esempio del fatto che anche lavorando sotto licenza GPL è possibile produrre dei risultati di qualità comparabile (in questo caso nettamente superiori) a quella dei prodotti commerciali.

Una delle cose che ho molto apprezzato è che, nonostante, si tratti di un lavoro esplicitamente scritto e distribuito secondo un modello libero, l'autore, allorché tratta l'argomento Freesoftware, non si lascia troppo trasportare dalla parte filosofica ma tenta di analizzare con una certa oggettività e senza "essere di parte" i vantaggi pratici di tale modello di sviluppo.

Tale approccio risulta molto gradito specialmente da chi desideri utilizzare il libro come supporto per corsi rivolti ad un'utenza aziendale.

Per contattare l'autore: daniele@pluto.linux.it

Per prelevare il libro: http://www.pluto.linux.it/ildp/AppuntiLinux

```
$ ls >/tmp/miopipe
```

Abbiamo realizzato una semplice comunicazione fra processi!

Vedremo nelle prossime puntate che la shell mette a disposizione un metodo molto potente per utilizzare gli unnamed pipe. Mediante l'operatore | è infatti possibile utilizzare l'output di un comando come input per un altro. L'esempio che segue ordina alfabeticamente le righe contenute in un file, mediante il comando sort, e mediante un pipe passa il risultato dell'elaborazione a uniq, il quale elimina le righe ripetute:

```
$ sort miofile | uniq
```

I socket

I socket sono una estensione del concetto di pipe ad una rete di calcolatori. Si tratta di una libreria di funzioni che offre dei metodi standard e molto semplici per far comunicare fra loro in modo bidirezionale due processi, anche funzionanti su macchine diverse. Per il trasporto dei dati vengono di solito utilizzati i protocolli TCP/IP, oppure un pipe se la comunicazione avviene fra processi funzionanti sulla stessa macchina. Esistono implementazioni dei socket che consentono di utilizzare anche altri metodi di trasporto, per esempio una connessione sopra il protocollo di rete IPX.

Dal punto di vista del programmatore, una volta aperta la connessione, il socket appare come un qualunque file, in cui si può scrivere e leggere mediante le consuete funzioni del linguaggio C (write, read, printf, scanf, ...). Ogni problematica relativa alla gestione dei protocolli di rete di più basso livello viene gestita direttamente dal kernel e dalla libreria.

I socket sono forse il metodo di comunicazione fra processi più utilizzato. Per rendersene conto basti pensare al fatto che ogni volta che preleviamo una pagina web viene gestito, mediante un socket, un canale di comunicazione fra il browser funzionante sulla nostra macchina ed il server web sul sito che stiamo navigando.

La memoria virtuale

Oltre alla semplice protezione della memoria, il kernel di Linux sfrutta i meccanismi messi a disposizione dalla MMU anche per offrire un'altra funzione interessante, la memoria virtuale, ovvero la possibilità di simulare su disco una uguale quantità di memoria RAM. Il funzionamento è concettualmente molto semplice ed è basato sulla gestione "a pagine" della memoria: quando il sistema necessita di allocare nuova memoria e non vi è più RAM disponibile, il kernel provvede a parcheggiare su disco le pagine di memoria riservate ad altri processi ma non utilizzate da un certo tempo (ad esempio perché il processo proprietario è sospeso oppure sta ciclando sulla stessa parte di codice). Nel momento in cui il legittimo proprietario tenterà di riaccedere alla pagina mancante, la MMU segnalerà al kernel la condizione d'errore e la routine di gestione della memoria si occuperà di ricaricare la pagina salvata su disco e di ripristinare il funzionamento del programma dal punto in cui è avvenuta l'interruzione. Anche se logicamente il programma vedrà la pagina nella posizione originaria, non è detto che fisicamente essa si trovi nello stesso posto di prima. Sarà perciò compito dell'hardware della macchina mappare le pagine di memoria in modo che logicamente vengano mostrate al processo come contigue, indipendentemente dalla reale posizione fisica in cui esse si trovano.

L'operazione di scambio delle pagine di memoria fra RAM e disco prende il nome di swapping (scambio) e si svolge in modo trasparente all'utente, che nota solamente il degrado delle prestazioni del sistema dovuto alla lentezza degli accessi al disco.

La memoria virtuale appare all'utente come se fosse memoria RAM e una macchina con 32 Mb di memoria RAM e 32 Mb di area di swap permette di fare girare le stesse applicazioni che possono essere utilizzate su una macchina con 64 Mb di RAM.

In Linux è possibile riservare all'utilizzo come memoria virtuale sia partizioni di disco che file residenti su un filesystem.

L'utilizzo di una intera partizione come "area di swap" consente prestazioni leggermente superiori rispetto all'utilizzo di un file, ma a scapito di dover decidere al momento dell'installazione del sistema la quantità di disco da usare come memoria virtuale. Utilizzando un file è invece possibile variare tale valore in qualunque momento. I due metodi possono comunque coesistere ed è possibile creare ed utilizzare contemporaneamente come memoria virtuale fino a 256 aree di disco, ognuna della dimensione massima di 256 Mbyte. Non essendoci una formula universalmente valida per determinare la quantità di spazio di swap necessario, la soluzione a mio avviso migliore è perciò quella di creare una partizione di swap di dimensioni opportune (stimando l'utilizzo che si farà del sistema) e di utilizzare i file di swap per aggiungere ulteriore memoria, qualora si renda necessaria.

Una partizione di swap deve essere creata in fase di installazione e viene identificata da un apposito identificativo (82=Linux Swap) nella tabella delle partizioni creata dal comando fdisk:

```
# fdisk
Using /dev/hda as default device!

Command (m for help): p

      Disk /dev/hda: 16 heads, 63 sectors, 4200 cylinders
      Units = cylinders of 1008 * 512 bytes

        Device Boot    Begin     Start       End    Blocks   Id  System
/dev/hda1   *         1         1      1023   515560+   6  DOS 16-bit >=32M
/dev/hda2          1024      1024      1836   409752   83  Linux native
/dev/hda3          1024      1837      1869    16632   82  Linux swap
/dev/hda4          1024      1870      4200  1174824   83  Linux native

Command (m for help):
```

Per creare un file di swap si utilizza invece il comando dd, che permette di copiare un certo numero di byte da un file ad un altro: nel nostro caso utilizzeremo come sorgente (if) il file speciale /dev/zero, che genera a getto continuo

caratteri con codice ASCII 0, e come destinazione (of) il file di swap che stiamo creando. I parametri bs e count indicano rispettivamente la dimensione ed il numero dei blocchi di dati da copiare.

```
# dd if=/dev/zero of=/mio_swap bs=1024 count=65536
```

In questo esempio abbiamo creato un file lungo 64 Mbyte (65536 blocchi da 1024 byte).

Una vola creato il file o la partizione, prima di poterla utilizzare come area di swap è necessario crearvi una opportuna struttura che consenta il salvataggio delle pagine di memoria. Ciò può essere fatto, sia nel caso delle partizioni che dei file di swap, mediante il comando mkswap, indicando come parametri il nome del file o partizione e la dimensione dell'area di swap da creare, espressa in blocchi da 1024k. L'opzione -c esegue il controllo dell'integrità del supporto magnetico.

```
# mkswap -c /mio_swap 65536
```

Nel caso si stia utilizzando una partizione, il comando diventa:

```
# mkswap -c /dev/hda3 16632
```

Per rendere attiva la nuova area di swap è sufficiente segnalarlo al kernel mediante il comando

```
# swapon /mio_swap
```

È possibile verificare che la nuova area sia stata aggiunta alla memoria virtuale della macchina richiedendo al sistema informazioni sulla memoria libera:

```
# free
             total       used       free     shared    buffers
Mem:         62972      61820       1152      12784      12948
Swap:        65533          0      65533
```

Si noti che parte della memoria viene utilizzata per contenere i buffer necessari al caching del filesystem. L'allocazione della memoria per i diversi scopi viene gestita in modo dinamico direttamente dal kernel, il quale, piuttosto che lasciarla inutilizzata, tende ad utilizzare per il caching del disco la maggior quantità possibile di memoria RAM, in modo da massimizzare le prestazioni del sistema. In caso di necessità tale memoria viene recuperata ed assegnata alle applicazioni che ne abbiano necessità (vedere Figura 1).

Le librerie condivise e altri metodi di risparmio delle risorse

Oltre alla memoria virtuale, Linux offre altri meccanismi che permettono di risparmiare risorse, i più importanti dei quali sono il demand loading (caricamento su richiesta) delle pagine di memoria e la condivisione del codice dei programmi (shared text e shared libraries).

Il meccanismo di demand loading permette di caricare in memoria solamente le pagine del codice di un programma che effettivamente vengono utilizzate e solamente nel momento in cui ne è richiesto l'uso. Esso permette risparmi notevoli nell'utilizzo della memoria, specialmente nel caso di programmi molto grandi di cui venga effettivamente utilizzata solamente una piccola parte del codice.

Il meccanismo sfrutta il principio che un programma non può essere cancellato dal disco su cui risiede mentre esso è in esecuzione. In questo modo la posizione di ogni parte del codice del programma è fissa su disco ed in caso di necessità il kernel è in grado di caricarla direttamente dal disco.

La condivisione del codice dei programmi eseguibili (shared text) è certamente uno dei metodi più importanti che Linux offre come ottimizzazione nell'utilizzo delle risorse. Essa consente di tenere in memoria solamente una copia del codice nel caso vengano lanciate più istanze dello stesso programma. Questa eventualità in un sistema multitasking è assai frequente e perciò tale soluzione comporta un notevole risparmio di memoria.

Una applicazione di questo meccanismo si trova nelle shared libraries (librerie condivise o librerie dinamiche). Per evitare di dover riscrivere lo stesso codice, i compilatori dei vari linguaggi di programmazione permettono l'utilizzo nei propri programmi di librerie, ovvero di raccolte di funzioni già pronte. Esse possono essere linkate (collegate) al codice del programma che le utilizza in fase di compilazione (librerie linkate in "modo statico"), oppure, per risparmiare spazio e per non portarsi dietro una copia inutile dello stesso codice con ogni singolo programma (la libreria standard del linguaggio C occupa circa 3 Mb), è possibile scegliere di linkare la libreria in "modo dinamico", ovvero a run-time. Sfruttando i meccanismi descritti precedentemente, il kernel è infatti in grado di caricare in memoria una singola copia del codice della libreria e di renderla disponibile ad ognuno dei programmi che ne necessitano. Così facendo, il codice del programma contiene solo dei riferimenti alle funzioni ma non il loro codice, e perciò occupa meno spazio su disco e in memoria. Le librerie pensate per funzionare in questo modo prendono il nome di shared libraries (librerie condivise) e sono contenute nella directory /lib. Esse comprendono un indice delle funzioni contenute al proprio interno, il quale rende possibile l'aggiornamento o la correzione di eventuali errori semplicemente sostituendo il file della libreria.

L'unico svantaggio delle shared libraries è dovuto al fatto che è necessario avere su disco oltre ai programmi anche una copia delle librerie che essi utilizzano. Il comando ldd permette di conoscere quali librerie sono richieste da un dato programma:

```
# ldd /bin/ls
libc.so.6 => /lib/libc.so.6 (0x40003000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x00000000)
```

Per permettere la compatibilità col vecchio software, di solito si preferisce tenere anche le versioni precedenti delle librerie ed utilizzare dei link simbolici per indicare la più recente:

```
/lib/libc.so.6 -> libc-2.1.1.so
/lib/libvga.so.1 -> libvga.so.1.1.7
```

Le librerie necessarie al funzionamento dei programmi fondamentali al boot del sistema non possono risiedere in /usr/lib in quanto questa directory potrebbe non essere disponibile al momento del boot della macchina (ad esempio perché essa risiede su un filesystem montato successivamente). Per

questo motivo si preferisce tenere nella directory /sbin la versione compilata in modo statico dei programmi più importanti.

La risoluzione delle richieste dei programmi di accedere alle funzioni contenute in una libreria viene effettuata dal linker dinamico ld.so, richiamato automaticamente al momento del caricamento del codice del programma.

Le librerie vengono cercate in alcuni percorsi standard ed altri possono essere aggiunti definendo in modo opportuno la variabile di sistema LD LIBRARY PATH (ad esempio modificando /etc/profile):

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/bin/rvplayer5.0
```

Aggiornamento delle shared libraries

Alle volte capita di provare ad eseguire sulla propria macchina un comando precompilato e di ricevere un messaggio d'errore, ad esempio:

```
incompatible library '/lib/libc.so.5.6.26' Require major version 6 and found 5
```

oppure

```
incompatible library '/lib/libc.so.6.5.23' Desire minor version >= 7 and found 5
```

Nel primo caso il programma non verrà eseguito, in quanto la versione della libreria, in questo caso 5, non corrisponde a quella minima richiesta. Nel secondo esempio invece l'incompatibilità è limitata alla sottoversione ed il programma ld.so tenta egualmente di utilizzare la libreria, anche se non si tratta di quella più aggiornata. L'utilizzo di una sottoversione di una libreria maggiore di quella richiesta (libc.so.6.x invece di libc.so.6.y) non causa di solito problemi (anzi, può servire per correggerne). Il cambio di versione (libc.so.7.x invece di libc.so.6.y) può invece causare malfunzionamenti in alcuni programmi, specialmente quelli che interagiscono con il sistema operativo (ps, w, top, ...), poiché ad una nuova versione della libreria può corrispondere qualche variazione importante anche a livello di kernel.

Un messaggio del tipo:

```
$ /usr/local/bin/mioprogramma
file not found
```

potrebbe anch'esso essere indice di un problema nelle librerie, ad esempio che si sta tentando di utilizzare un programma che fa uso di vecchie librerie di tipo a.out su un sistema recente che utilizza il formato ELF, o viceversa. Un'altra causa di problemi è il fatto che i programmi più recenti utilizzano la versione GNU della libreria del linguaggio C (glibc), al contrario di quelli precedenti che usavano la libc 5.

Per aggiornare il link simbolico che punta alla versione più recente di una libreria è necessario utilizzare un comando simile al seguente:

```
ln -sf /lib/libc.so.6 -> libc-2.1.3.so
```

Non è consigliabile cancellare la vecchia libreria e poi ricreare il link, in quanto il comando ln necessario per compiere quest'ultima operazione potrebbe a sua volta dipendere dalla libreria appena cancellata.

Conclusioni

Anche per questo mese abbiamo terminato. La prossima volta parleremo della shell, l'interprete di comandi di Linux.

MC

Esplora con noi

il mondo dell'informatica!

Vuoi esplorare il pianeta "INFORMATICA" e viaggiare sicuro? Cerchi delle guide esperte per scoprire nuove frontiere? Noi facciamo al caso tuo.
Forti dell'esperienza pluriennale di MCmicrocomputer, Byte Italia, WoW World of Web e molte altre riviste del settore, lo staff Pluricom è a tua disposizione con corsi a tutto campo nella sua struttura di formazione.

CORSI DI BASE

ECDL

1) Concetti teorici di base dell'Information Technology (Basic concepts)
2) Gestione dei documenti (File management and O.S.)
3) Elaborazione dei testi (Word processing)
4) Fogli elettronici (Spreadsheet)
5) Basi di dati (Database)
6) Strumenti di Presentazione
7) Internet

Sono aperte le iscrizioni

Altri corsi

Corsi su misura per aziende

- Teoria della grafica - Elaboratore di immagini digitali - Operatore multimediale
- Java
- Front Page - Internet per specifiche categorie professionali
- SQL Filemaker Acces

INIZIATIVE SPECIALI
Corso di Internet per psicologi, psichiatri e pedagoghi
Corso multimediale per patenti nautiche
Esclusiva nazionale

La Pluricom - MCmicrocomputer School è Test Center per il rilascio dell'ECDL. I corsi sono strutturati sulla base dei contenuti previsti dal programma ECDL per il conseguimento della patente informatica europea.

ECDL

European Computer Driving Licence

Pluricom
INFORMAZIONE e FORMAZIONE

Pluricom S.r.l. - Viale Ettore Franceschini, 73 - 00155 Roma - Tel.06 43219.312 - fax 06 43219.301 - e-mail: corsi@pluricom.it