Sistemi Operativi

aspetti teorici, applicazioni pratiche in Linux

2012.02

Indice

1. Premessa e prerequisiti

Lo scopo di questi appunti è quello di trattare alcuni aspetti teorici dei sistemi operativi e, nel contempo, fornire un riscontro pratico con aspetti amministrativi. In pratica in ogni paragrafo vengono affrontate problematiche generali della progettazione di un sistema operativo, laddove nei suoi sotto-paragrafi si tratta lo stesso argomento attraverso l'esame di alcuni comandi, soluzioni pratiche e caratteristiche implementative del sistema operativo Linux. Le due trattazioni procedono, quindi, in parallelo.

Per una maggiore comprensione degli argomenti trattati, è necessaria una conoscenza dei principi generali del funzionamento di un computer e della sue componenti. Per poter provare le applicazioni pratiche, e avere maggiori possibilità di memorizzarle, nel computer deve essere installato Linux o è necessario, almeno, dotarsi di un live-CD. La particolare distribuzione, in linea generale, non ha importanza poiché si farà riferimento alle caratteristiche di Linux, presenti in qualunque distribuzione che contiene tale nome al suo interno. Tuttavia qualche directory di configurazione potrebbe, in qualche distribuzione, avere un'allocazione diversa da quella riportata in queste note: la distribuzione di riferimento in questi appunti è Ubuntu.

È inoltre richiesta una conoscenza base dei comandi della bash e del sistema Linux in generale (fare riferimento a BasicLinux).

Negli esempi concreti sono utilizzate le convenzioni evidenziate di seguito:

```
$ ls
bfsh-koc.zip  duex              prova      scan2.sxw    uno.c
Blowf.c       Esempio_A01.java  prova1     stringhe     unox
Blowf.h       Main.c            prova.cpp  stinghe.cpp
due.cc        prova.s           storia1
```

i caratteri in **grassetto** rappresentano l'eco su video dei ciò che è digitato da tastiera (in questo caso il comando `ls` per visualizzare l'elenco dei file contenuti nella directory di lavoro), il resto in caratteri `normali` rappresenta la risposta del sistema al comando impartito.

In questi appunti si esamineranno molto spesso file di configurazione del sistema. Si tratta di file in formato testo:

```
# Remove home directory and mail spool when user is removed
REMOVE_HOME = 0

# Remove all files on the system owned by the user to be removed
REMOVE_ALL_FILES = 0

# Backup files before removing them. This options has only an effect if
# REMOVE_HOME or REMOVE_ALL_FILES is set.
BACKUP = 0

# target directory for the backup file
BACKUP_TO = "."
```

I file di configurazione di Linux sono abbondantemente commentati. Le righe precedute dal simbolo `#` sono commenti. Tutte le opzioni possibili sono presenti e se una opzione non si vuole rendere attiva, semplicemente, viene anteposto il simbolo `#` e diventa un commento, se si vuole riabilitarla successivamente basta togliere il simbolo di commento.

Una cosa importante da ricordare è che, spesso, nel corso di queste note verranno utilizzati comandi

impartiti con diritti di super-utente. In questi casi si ha accesso completo a tutte le risorse di sistema e si possono impartire comandi il cui uso improprio può causare gravi malfunzionamenti. Occorre porre estrema attenzione nell'uso di tali comandi ed essere sicuri di quello che si sta facendo. Anche se tutti comandi riportati sono stati testati l'autore di queste note non si assume alcuna responsabilità per l'uso improprio ed eventuali danni provocati dalle istruzioni riportate nelle note stesse.

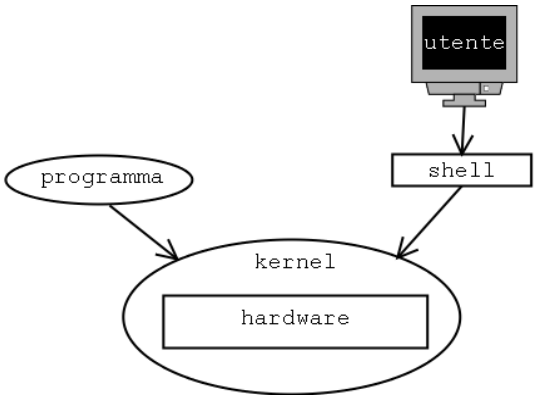
2. Che cosa è un Sistema Operativo?

Non esiste una definizione univoca di questo insieme di moduli software che pure rappresentano una presenza costante e discreta con cui qualsiasi utente di computer ha avuto a che fare. D'altra parte gestire un sistema di elaborazione senza l'aiuto di questo software, così come era la situazione nella *preistoria* dei computer, non sarebbe cosa agevole richiedendo un insieme di conoscenze tali da superare quelle di un solo individuo. E questo a maggior ragione quando si tratta di gestire la complessità di un computer dei nostri giorni.

In linea generale si può dire che il sistema operativo assolve a due funzioni:

- **Presentare una *macchina virtuale* più facile da programmare dell'hardware sottostante**. Il linguaggio macchina, l'unico effettivamente comprensibile da un computer, fornisce istruzioni primitive e complesse da utilizzare. Per poter effettuare anche semplici operazioni è necessario tenere conto di tutti i dettagli di funzionamento delle parti interessate. Si pensi, per esempio, a cosa può voler dire rintracciare un insieme di dati in una memoria di massa: bisogna pensare a mettere in movimento la meccanica del disco, aspettare che raggiunga la velocità giusta, conoscere il numero dei blocchi in cui è suddiviso il disco, il numero delle tracce e dei settori, il modo in cui i dati sono conservati, ... Ammesso che si riesca una volta per tutte a esprimere in maniera corretta tutti i parametri che servono, c'è da aggiungere anche un altra osservazione: se si vuole utilizzare una stampante, ovviamente, bisogna cambiare tutto non appena la si sostituisce con un nuovo modello. Chiaramente è molto più pratico e agevole avere a che fare con una interfaccia ad alto livello che, nel caso degli esempi cui si è accennato prima, faccia vedere le informazioni conservate su disco come un insieme di file ognuno con il proprio nome e che si possano utilizzare istruzioni del tipo *metti a disposizione quel tale file*, *leggi* o *scrivi*, *chiudi il file*, oppure *manda in stampa i dati* senza preoccuparsi della posizione delle testine di lettura-scrittura del drive per dischi o delle specifiche tecniche del tipo di stampante: si tratti di una inkjet o di una laser, ci si deve preoccupare solo dei dati da stampare. In definitiva il compito del sistema operativo è quello di nascondere le complessità della macchina, fornire una macchina virtuale più semplice da programmare e una interfaccia amichevole.
- **Gestire le risorse hardware**. Per espletare le funzioni richieste da un programma è necessario che tutte le risorse di un sistema di elaborazione agiscano in maniera coordinata. Si pensi ad una orchestra che deve riprodurre un brano musicale: se, per esempio, la sezione degli archi non agisce in maniera coordinata con la sezione degli ottoni non si avrà come risultato niente che assomigli a qualcosa di piacevole per l'udito. Per restare nell'ambito dell'esempio proposto si può affermare che il direttore (nel mondo del computer il sistema operativo) assume il ruolo fondamentale di coordinatore delle risorse disponibili: nel caso dell'esempio le diverse sezioni dell'orchestra, nel caso del sistema di elaborazione tutte le apparecchiature hardware disponibili. Oltre a coordinare l'intervento delle varie parti, il sistema operativo deve pure provvedere ad attribuire le varie risorse in maniera ordinata. Si pensi, a titolo di esempio, a due programmi che necessitano dell'uso della stampante. In questo caso il sistema operativo dovrà provvedere a fare

in modo che le uscite di un programma non si mischino in modo caotico con le uscite dell'altro programma.



Alla base di un sistema operativo c'è il *kernel*, un insieme di programmi che controllano l'accesso, allocano le risorse, gestiscono l'input e l'output, la memoria centrale, stampanti e quanto altro faccia parte dell'insieme delle risorse disponibili. È la parte del sistema operativo cui si rivolgono i programmi per le loro esigenze. I programmi non accedono direttamente alle varie componenti hardware ma hanno a disposizione una serie di dispositivi logici messi a disposizione dal kernel.

La *shell* è un programma che si interfaccia con l'utente: ne interpreta le richieste, permette l'esecuzione dei programmi richiesti. È un programma intermediario fra utente e kernel.

I *programmi di utilità* sono un insieme di programmi per la manutenzione del sistema, l'editing di testi ecc.

2.1 Rilevamento di periferiche in Linux

Per poter permettere l'interazione fra i programmi e qualsiasi tipo di periferica al kernel si affiancano moduli software (***driver***) per la gestione delle periferiche in modo che esse siano trasparenti ai programmi: un programma scrive su una periferica al di là delle sue specifiche tecniche, sia essa, per esempio, un Hard Disk o una chiavetta USB.

Esistono due modi con cui un kernel può interagire con i driver:

- **modello monolitico**: il kernel è un unico programma che contiene al suo interno tutti i driver per governare le periferiche. Tutti i driver girano assieme al kernel (in *kernel-space*) con il massimo dell'efficienza possibile
- **modello microkernel**: il kernel è il più piccolo possibile e i driver girano (in *user-space*) come programmi qualsiasi

Nel kernel di Linux, originariamente sviluppato secondo il modello monolitico, i moduli per la gestione delle periferiche si possono caricare dinamicamente quando servono o, in alternativa, si possono includere nel kernel ricompilandoli con esso.

Dal punto di vista dell'utente si nota che una volta inserita una periferica, il sistema provvede a installarla in modo automatico. L'automatismo si basa su tre componenti:

1. il **kernel**: utilizza la directory virtuale `/sys` i cui file rappresentano informazioni sul kernel e non sono dati effettivamente esistenti nell'HD. Ogni device esistente nel sistema ha una directory nella `/sys` contenete file con informazioni sulle risorse allocate
2. il demone **Udev**: programma in esecuzione che non interagisce con l'utente (*demone*) che controlla la directory `/sys`, carica i moduli necessari alla gestione delle periferiche, genera un file associato ad ogni periferica nella `/dev` e notifica ad HAL la eventuale presenza di nuovi

device

3. **HAL** (Hardware Abstraction Layer: strato di astrazione sull'hardware): gestisce un archivio delle periferiche e funziona da interfaccia fra queste e le applicazioni. Le applicazioni ricevono da HAL notifiche sulla presenza di nuove periferiche e interrogano HAL per conoscerne le caratteristiche.

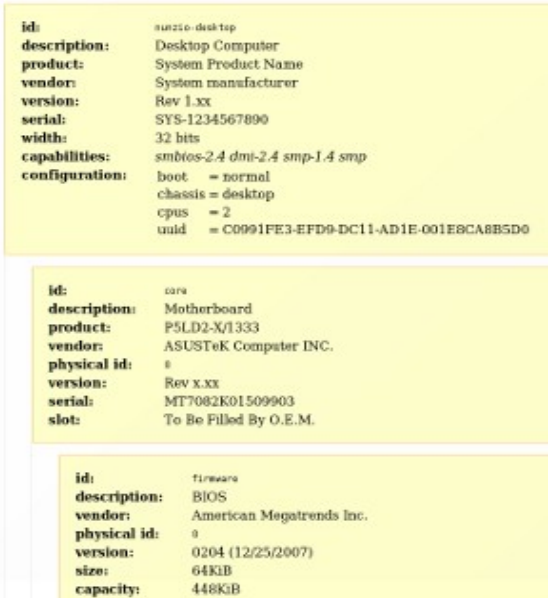
L'utente può ottenere informazioni sull'hardware interrogando i file di log del kernel ovvero i file in cui il kernel annota il resoconto delle proprie attività (il diario di bordo del sistema).

```
$ dmesg | tail
[   20.660198]   groups: 1 0
[   22.948470] CPU0 attaching NULL sched-domain.
[   22.948476] CPU1 attaching NULL sched-domain.
[   22.968636] CPU0 attaching sched-domain:
[   22.968640]  domain 0: span 0-1 level MC
[   22.968642]   groups: 0 1
[   22.968646] CPU1 attaching sched-domain:
[   22.968648]  domain 0: span 0-1 level MC
[   22.968650]   groups: 1 0
[   30.304019] eth0: no IPv6 routers present
$ dmesg | tail
[ 4433.851997] scsi 4:0:0:0: Direct-Access     SanDisk  Cruzer Mini      0.2  PQ:
0 ANSI: 2
[ 4433.852857] sd 4:0:0:0: Attached scsi generic sg2 type 0
[ 4433.855604] sd 4:0:0:0: [sdb] 250879 512-byte logical blocks: (128 MB/122 MiB)
[ 4433.856353] sd 4:0:0:0: [sdb] Write Protect is off
[ 4433.856359] sd 4:0:0:0: [sdb] Mode Sense: 03 00 00 00
[ 4433.856363] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 4433.865863] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 4433.865874]  sdb: sdb1
[ 4433.870844] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 4433.870852] sd 4:0:0:0: [sdb] Attached SCSI removable disk
```

Il comando `dmesg` consente la visualizzazione dei log del kernel. Nell'esempio è utilizzato (concatenato) con `tail` per visualizzarne le ultime righe. I file di log del sistema sono parecchio lunghi e qui interessavano le ultime righe scritte dal kernel dopo l'introduzione di una chiavetta USB.

Il primo output del comando è quello che si ottiene lanciando il comando prima di inserire la chiavetta, il secondo output è il risultato dello stesso comando ottenuto dopo l'introduzione della chiavetta. Dall'esame degli output si nota la marca della chiavetta e il device che il kernel associa ad essa (`sdb`). Il device può essere utilizzato per montare la chiavetta ed accedere ai dati contenuti in essa.

2.2 Informazioni sull'hardware



Alcune delle informazioni sull'hardware installato, conservate dal kernel nella `/sys`, possono essere visualizzate con i comandi:

- `lscpu` per avere informazioni sulla CPU e sulla sua architettura
- `lspci` per i dispositivi che comunicano su bus PCI
- `lsusb` per i dispositivi che comunicano su bus USB
- `lshw` per informazioni su tutto l'hardware installato. Esiste la possibilità di formattare l'output per una visualizzazione più comoda e comprensibile:

```
$ lshw -html > mioHardware.html
```

Usato nella maniera suddetta il comando restituisce il proprio output in formato HTML. L'operatore `>` direziona l'output al file specificato invece che al monitor. Il file può essere infine visualizzato utilizzando un browser.

2.3 Informazioni sulle risorse occupate: /proc

La `/proc` è una *pseudo directory* nel senso che, se si esegue il comando `ls -l` si otterrà un elenco di file con dimensione nulla. In realtà non c'è nulla di strano perché i file mostrano informazioni su vari aspetti del sistema in esecuzione. La maggior parte dei file presenti in questa directory sono visualizzabili per mezzo del comando `less`: si tratta infatti di file di testo che, trattando situazioni in continuo divenire (per esempio l'occupazione di memoria), sono generati dal sistema quando deve fornire una risposta alla richiesta di visualizzazione, e una successiva ripetizione dello stesso comando fornirà dati diversi. In definitiva quando il sistema deve rispondere al comando di visualizzazione, scatta una fotografia dello stato della risorsa e fornisce le informazioni. Di conseguenza alla prossima richiesta viene scattata una nuova fotografia che sarà diversa della prima essendo nel frattempo cambiato lo stato della risorsa.

Verranno esaminate adesso alcune risorse disponibili nel sistema:

```
$ cd /proc
$ less cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 8
model name      : Pentium III (Coppermine)
stepping        : 6
cpu MHz         : 800.024
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
```

```
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat
pse36 mmx fxsr sse
bogomips        : 1582.76
```

nell'esempio proposto, dopo aver selezionato la directory `/proc`, si è richiesta la visualizzazione delle informazioni sulla CPU presente nel sistema. Nel caso in esame si tratta di un Intel Pentium III a 800 Mhz.

Allo stesso modo si potrebbe voler avere informazioni sulle porte di I/O presenti:

```
$ less ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
...
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)
...
0378-037a : parport0
037b-037f : parport0
03c0-03df : vga+
...
```

o sulla mappatura della memoria:

```
$ less iomem
00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-13feffff : System RAM
  00100000-0024b8dd : Kernel code
0024b8de-003457a3 : Kernel data
...
```

Le informazioni sull'occupazione di memoria sono accessibili visualizzando lo pseudo-file `meminfo`:

```
$ less /proc/meminfo
MemTotal:       320424 kB
MemFree:         56112 kB
Buffers:          9576 kB
Cached:         176320 kB
SwapCached:          0 kB
Active:         122792 kB
Inactive:       124780 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       320424 kB
LowFree:         56112 kB
SwapTotal:      787144 kB
```

```
SwapFree:       787144 kB
Dirty:              40 kB
Writeback:           0 kB
Mapped:         129288 kB
Slab:            12432 kB
CommitLimit:    947356 kB
Committed_AS:   126420 kB
PageTables:       1160 kB
VmallocTotal:   704504 kB
VmallocUsed:      5300 kB
VmallocChunk:   697536 kB
HugePages_Total:     0
HugePages_Free:      0
Hugepagesize:     4096 kB
```

i valori vengono dettagliati per ogni voce presente.

```
$ free
             total       used       free     shared    buffers     cached
Mem:        320424     264256      56168          0       9588     176456
-/+ buffers/cache:      78212     242212
Swap:       787144          0     787144
```

Il comando `free`, facendo riferimento alle stesse informazioni, ne fornisce una visione sintetica.

3. Le interfacce utente

Il problema dell'interfacciamento con un elaboratore è stato sempre uno dei punti su cui si sono concentrati gli sforzi dei ricercatori. La forma più classica di shell è la riga di comando: la shell, per mezzo di un apposita stringa di invito (il *prompt*) avvisa che è pronta a ricevere una riga di comando da tastiera. La shell può fornire all'utente, oltre al semplice interprete della linea di comando, delle estensioni che la portano ad avere le caratteristiche di un linguaggio di programmazione con la possibilità di scrivere dei veri e propri programmi (gli *script* di shell), per esempio per automatizzare operazioni ripetitive che, una volta lanciati, vengono interpretati dalla shell in modalità *batch*: i comandi compresi nello script sono avviati in sequenza senza intervento dell'operatore.

La diffusione sempre maggiore dei computer allarga la base dei suoi utilizzatori. Se la shell a riga di comando va senz'altro bene per l'amministratore del sistema (la persona che si occupa della manutenzione, in tutte le sue forme, del sistema stesso) e anzi permette di eseguire le operazioni necessarie in modo efficiente e rapido (si può affermare che è proprio questo lo scopo della shell), tuttavia richiede conoscenze del sistema non banali. Un utente che abbia la necessità di usare il computer, per esempio, per l'elaborazione di testi, è interessato a rendersi produttivo nel più breve tempo possibile e, d'altra parte, la presenza di sole shell a caratteri escluderebbe la possibilità di utilizzo del computer da parte di una ampia fascia di utenti. È questo il motivo che ha spinto i ricercatori a progettare shell che, almeno, consentissero di effettuare le operazioni minime in modo quanto più semplice possibile, al limite senza richiedere alcuna conoscenza specifica da parte dell'utente. Sono quelle shell che oggi vengono chiamate *sistemi desktop* perché il principio su cui si basano è quello della cosiddetta *metafora della scrivania*: l'obiettivo è quello di mettere l'utente davanti ad un ambiente familiare e intuitivo utilizzando sistemi che fanno uso di rappresentazioni grafiche, icone per rappresentare applicativi e che permettano di interagire con esse per mezzo di un uso massiccio di periferiche di puntamento (per esempio il mouse).

> *Un tempo l'interfaccia utente era l'ultima parte del sistema a essere progettata. Oggi è la prima. Ci si è resi conto che essa è di primaria importanza perché, tanto per i principianti quanto per i professionisti, ciò che si presenta ai sensi di una persona* ***è*** *il calcolatore di quella persona.* ***L'illusione utente****, così come i miei colleghi ed io la battezzammo al Palo Alto Research Center della Xerox, è il mito semplificato che ognuno costruisce per cercare di spiegare le azioni del sistema e ciò che dovrebbe fare poi. Molti dei princìpi e dei dispositivi sviluppati per migliorare quell'illusione sono diventati un luogo comune della progettazione del software. Forse il principio più importante è WYSIWYG (What You See Is What You Get): l'immagine sullo schermo è sempre una fedele rappresentazione dell'illusione dell'utente. La manipolazione dell'immagine in un certo modo produce immediatamente qualcosa di prevedibile sullo stato della macchina (così come l'utente immagina quello stato). (Alan Kay)*

Nei laboratori del MIT (Massachusetts Institute of Technology) fu implementato, per la prima volta nel 1984, X Window System, un sistema grafico per ambienti Unix. L'obiettivo era appunto quello di creare una interfaccia che rendesse più facile l'uso dei calcolatori. Ai tempi si avevano ancora display grafici monocromatici e le potenze di calcolo erano esigue. Nel 1988 nasce l'X Consortium associazione composta prevalentemente da ricercatori del MIT e nascono le prime implementazioni del sistema grafico X. Il sistema diventato standard fu chiamato X386 e veniva utilizzato sulle macchine delle aziende partner del progetto: IBM, Sun, HP.

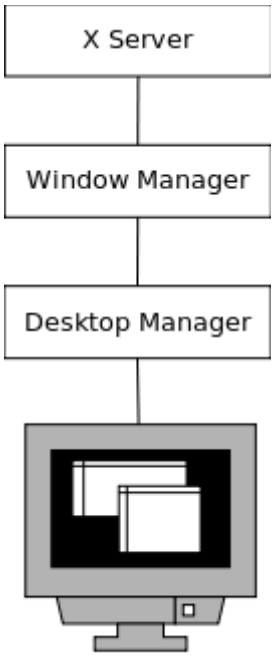
3.1 Il sistema grafico di Linux

Tra il 1991 e il 1992 un gruppo di ricercatori si stacca da X Consortium, effettuando quello che nel gergo di Linux viene chiamato *fork*. In pratica ad un certo punto la strada dello sviluppo del progetto, che prima era unica, si biforca. Il nuovo progetto che venne chiamato Xfree86 si prefiggeva, in maniera prevalente, lo scopo dell'adattamento del X Window System ai sistemi con architettura Intel. In tempi recenti lo sviluppo di X come software libero è curato da X.Org Foundation.

In alcuni sistemi operativi l'ambiente grafico e il sistema operativo rappresentano un blocco monolitico: le due cose sono inscindibili, l'unica interazione con il sistema operativo avviene per mezzo della shell grafica e non è possibile fare ciò che l'ambiente grafico non permette. Nel caso di Linux invece ci si trova davanti ad un sistema modulare fortemente personalizzabile: si tratta di un *abito* che *può* venire indossato *sopra* il sistema stesso e *cambiato* come e quando si vuole.

X11, come viene chiamato oggi facendo riferimento allo standard definito dalla fondazione, è un sistema basato sul modello client/server. Un programma server fornisce determinati servizi ad uno o più programmi client che ne fanno richiesta. Client e server del sistema X11 comunicano attraverso l'X Protocol, un insieme di regole che governano il traffico di informazioni fra il server e i vari client.

Più in particolare il sistema grafico di Linux è composto da:



- **Server grafico X**. Le funzioni svolte dal server sono minime: si occupa soltanto di eseguire le richieste di disegno da parte dei client, della gestione del mouse e dell'hardware del video.
- **Window manager**. È il client che gestisce la decorazione della finestra in cui gira il programma e le proprietà delle finestre stesse.
- **Desktop manager**. È il client che si occupa di fornire, per esempio, i pannelli per il lancio di applicazioni, file manager per la navigazione nel file system. È l'interfaccia più esterna e quella con cui è a contatto l'utente. Si tratta di client personalizzabili e sostituibili in qualsiasi momento. Si va da quelli più complessi e pesanti, dal punto di vista delle occupazioni di risorse, che hanno l'obiettivo di fornire un sistema *chiavi in mano* con acclusi programmi specifici sviluppati per l'ambiente, come Gnome e KDE che sono quelli più noti, a quelli meno avidi di risorse come XFce, WindowMaker o Enligthenment.

L'impostazione a moduli distinti ha permesso di utilizzare window manager con effetti avanzati 3D modificando solo marginalmente il server X.

3.2 Il modello client/server di X11

logo di X Window System

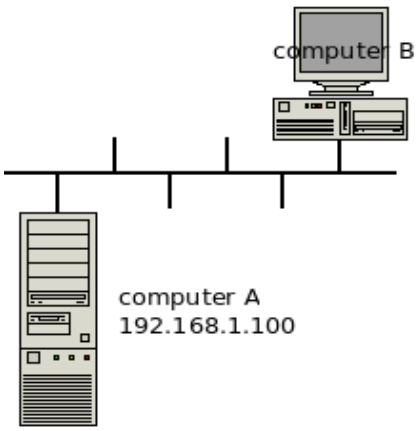
In accordo al modello client/server un server X gira in un computer con un display grafico, comunica e fornisce servizi a vari client, solo che nel sistema client/server di X i rapporti sono, dal punto di vista dell'utente, come capovolti rispetto al modello standard dove l'utente utilizza un client per richiedere servizi ad un server che può girare anche in un computer remoto. In X il server gira nel computer dell'utente mentre i client possono girare anche in computer diversi.

Il server agisce da tramite tra l'utente e il programma client:

- accetta richieste di output grafico (finestre) dal programma client
- mostra gli output del programma all'utente, riceve gli input (per mezzo di tastiera e mouse) dall'utente e li trasmette al programma client.

Le comunicazioni fra il server e i client avvengono attraverso un canale di rete e, inoltre, il server e il client possono girare sulla stessa macchina.

L'aderenza al modello C/S di X11 permette, all'interno di una rete, di utilizzare computer anche obsoleti con prestazioni ridotte, anche senza HD, per far girare programmi grafici notoriamente avidi di risorse, che in realtà gireranno su un computer potente della rete, ma mostreranno il loro output e riceveranno i controlli dell'utente tramite tastiera o mouse dai computer obsoleti.



Premesse:

- Nel/nei computer B è configurato ed in esecuzione il server X. Anche se questi computer non hanno HD, se sono dotati di un lettore di CD-ROM, possono far girare un live-CD con il server già configurato (ce ne sono di adatti anche a computer molto datati basta cercare, per esempio, in http://distrowatch.com).
- Il computer A si occuperà di far girare gli applicativi (i client) che mostreranno gli output sui computer B. il computer A deve avere potenze di calcolo adeguate, e inoltre mette a disposizione lo spazio nel proprio HD. Nel computer A sono presenti gli account per gli utenti che si collegheranno dai computer B. Nel computer è attivo il server ssh.

Nel computer B si apre un terminale e si lancia la connessione per far girare applicazioni grafiche in remoto:

```
$ ssh -X tux@192.168.1.100
```

- `ssh` (Secure SHell) apre un canale criptato per il collegamento
- l'opzione `-X` permette di far girare applicazioni grafiche in remoto
- `tux` è l'identificativo di un utente che ha l'account nel computer A
- `192.168.1.100` è infine l'indirizzo IP del computer A su cui gireranno i client

Effettuato il collegamento e dopo aver inserito la password dell'utente `tux` si può avviare, digitando la stringa, qualsiasi applicativo esattamente come se si fosse sul computer A solo che le interazioni avvengono sul computer B: i programmi rispondono alla tastiera e al mouse dell'utente che lavora sul computer B che ne vede gli output sul proprio monitor.

4. Storia dei sistemi operativi

Anche se *"in ogni storia non vi è mai un punto iniziale prima del quale non sia accaduto alcun fatto di rilievo e dopo il quale esso invece si sia verificato"*, si può considerare la *macchina analitica* del matematico inglese Charles Babbage (1792-1871) il capostipite dei calcolatori . Anche se Babbage spese la sua vita e i suoi averi nel tentativo di costruzione della sua macchina, in realtà questa non vide mai la luce. La tecnologia del tempo non era in condizioni di costruire ruote ed ingranaggi con quella precisione richiesta. Accanto al nome di Babbage va ricordato anche quello di Lady Ada Augusta Byron Contessa di Lovelace che comprese le enormi capacità della macchina analitica e propose alcuni programmi: è considerata la prima programmatrice della storia.

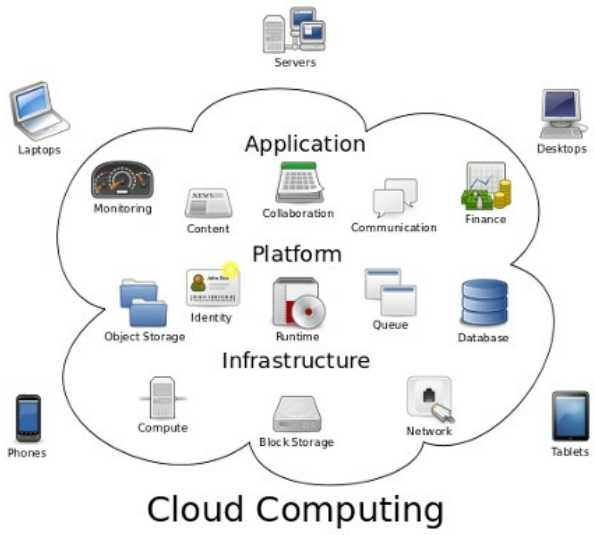
- **La prima generazione** (valvole e piastre 1945-1955). Intorno alla metà del 1940 vennero costruite le prime macchine calcolatrici. In diverse parti del mondo gli scienziati, spinti anche da esigenze di utilizzi bellici, furono in grado di far diventare realtà gli studi di Babbage. Alcuni nomi sono ricordati ancora oggi: John von Neumann a Princeton, Presper Eckert all'Università della Pennsylvania, Konrad Zuse in Germania. Le macchine di cui si parla, ovviamente, erano molto diverse da come li conosciamo oggi. La tecnologia costruttiva principale prevedeva l'uso

di valvole e le macchine che le usavano erano enormi, riempivano intere stanze e nonostante le vertiginose velocità riferite ai tempi, erano molto più lente del più piccolo dei calcolatori odierni. Non esistevano linguaggi di programmazione o sistemi operativi e la programmazione veniva fatta direttamente in linguaggio macchina o, più spesso, effettuando collegamenti su schede e controllando le funzioni della macchina. Un passo avanti si ebbe all'inizio del 1950 con l'introduzione delle schede perforate che permettevano di scrivere i programmi su schede che il calcolatore poteva leggere.

- **La seconda generazione** (transistor e sistemi batch 1955-1965). A metà degli anni '50 furono introdotti i transistor e i calcolatori divennero più affidabili. Solo poche grosse industrie o università erano in grado di affrontare il costo delle macchine del tempo. I *job* (lavori) erano eseguiti uno di seguito all'altro: un operatore si occupava di caricare in macchina le schede con perforate le istruzioni da eseguire, ritirare il risultato dalla stampante e procedere con il prossimo lavoro. Ci si rese conto che si sprecava molto tempo macchina in attesa che l'operatore provvedesse al carico e scarico dei lavori e, poiché il tempo macchina era una risorsa estremamente costosa, bisognava cercare una soluzione diversa. La soluzione trovata fu il *sistema batch*. In pratica si collezionavano una serie di job, venivano caricati su un nastro magnetico usando un piccolo calcolatore poco costoso adatto per caricare schede, ma poco per i calcoli numerici. Raccolti i job, l'operatore caricava in macchina un programma (l'antenato del sistema operativo) che leggeva dal nastro e mandava in esecuzione i programmi uno di seguito all'altro. I risultati venivano scaricati su un nastro che veniva portato sul calcolatore piccolo per la stampa *off-line* (scollegata dal calcolatore principale).
- **La terza generazione** (circuiti integrati e multiprogrammazione 1965-1980). Un ulteriore passo avanti nelle prestazioni e nella miniaturizzazione dei calcolatori si ebbe con l'introduzione dei circuiti integrati: i calcolatori si diffondono e i sistemi operativi cominciano ad avere il ruolo e la diffusione fra gli utenti che oggi conosciamo, utilizzando soluzioni tecniche anche sconosciute nella generazione precedente. È in questo periodo che si afferma la *multiprogrammazione*. In precedenza se un job si fermava in attesa di operazioni di I/O, la CPU restava inattiva. Se vengono effettuate elaborazioni commerciali che richiedono molte operazioni di I/O, l'inattività della CPU può anche essere quantificabile con percentuali del tipo 80%, 90% sul totale del tempo macchina. La soluzione adottata richiede il partizionamento della memoria in modo da contenere più job in contemporanea. La CPU divide il suo tempo, a turno, fra i vari job: si tratta dei sistemi a suddivisione di tempo (*time-sharing*). Avere molti job contemporaneamente in memoria richiede inoltre la presenza di hardware speciale per la protezione dalle intromissioni e dai danni da parte di altri job.
- **La quarta generazione** (circuiti a larga scala di integrazione e personal computer 1980-oggi). Con l'introduzione della larga scala di integrazione che permetteva di stipare in un chip di un centimetro quadro migliaia di transistor, le dimensioni dei calcolatori diminuiscono ulteriormente, l'introduzione del microprocessore permette inoltre di concentrare in un unico circuito le funzioni della CPU. I costi dell'hardware diminuiscono in maniera notevole e così anche un singolo individuo può avere il proprio calcolatore personale ed usufruire delle potenze di calcolo di un sistema di elaborazione. Comincia a diffondersi il concetto di *software user-friendly*, software orientato agli utenti che non sanno niente di computer, che non hanno alcuna intenzione di imparare qualcosa, ma vogliono solo utilizzare la macchina per far crescere la propria produttività. Le moderne linee direttive vanno verso la direzione di sistemi con caratteristiche sempre più multimediali e sistemi operativi di rete e distribuiti. In un sistema di

rete un utente con la propria macchina e il proprio sistema operativo può connettersi ad una macchina remota per condividere risorse. In un sistema distribuito tutto appare come se fosse presente in un unica macchina quando in realtà si utilizzano, in maniera trasparente (l'utente non ha percezione di ciò), risorse distribuite in macchine diverse: un programma può girare in una macchina, i file risiedere in un altra macchina.

Una tecnologia che tende a utilizzare queste ultime caratteristiche descritte è il cosiddetto **cloud computing**:



In informatica con il termine inglese cloud computing (in italiano nuvola informatica) si indica un insieme di tecnologie che permettono, tipicamente sotto forma di un servizio offerto da un provider al cliente, di memorizzare/archiviare e/o elaborare dati (tramite CPU o software) grazie all'utilizzo di risorse hardware/ software distribuite e virtualizzate in Rete.

...

È noto come, utilizzando varie tipologie di unità di elaborazione (CPU), memorie di massa fisse o mobili come ram, dischi rigidi interni o esterni, Cd/DVD, chiavi USB, eccetera, un computer sia in grado di elaborare, archiviare, recuperare programmi e dati. Nel caso di computer collegati in rete locale (lan) o geografica (wan) la possibilità di elaborazione/ archiviazione/recupero può essere estesa ad altri computer e dispositivi remoti dislocati sulla rete stessa.

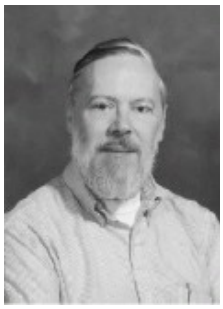
Sfruttando la tecnologia del cloud computing gli utenti collegati ad un cloud provider possono svolgere tutte queste mansioni, anche tramite un semplice internet browser. Possono, ad esempio, utilizzare software remoti non direttamente installati sul proprio computer e salvare dati su memorie di massa on-line predisposte dal provider stesso (sfruttando sia reti via cavo che senza fili).

(testo e grafico da Wikipedia)

Queste linee evolutive sono molto controverse e provocano diverse riserve a causa di notevoli rischi cui possono essere esposti gli utenti che le utilizzano: da problemi di sicurezza e privacy (i dati personali anche sensibili, risiedono in posti poco controllabili dall'utente e possibile preda di accessi non autorizzati e spionaggio industriale nel caso di dati aziendali), a problemi politici (dati residenti in paesi diversi da quello in cui risiede l'utente possono non essere sempre disponibili) per finire a problematiche tipicamente informatiche (non accessibilità dei dati in conseguenza a non continuità o disfunzioni del servizio, difficoltà di migrazione dei dati da un provider ad un altro).

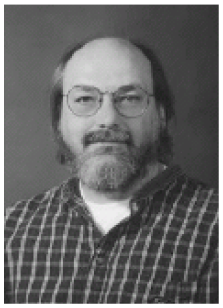
4.1 Da Unix a Linux: lo sviluppo a bazar

Nel 1965 un gruppo di ricercatori dei Bell Laboratories (la compagnia telefonica degli Stati Uniti), della General Electric (grossa azienda che, fra l'altro, costruiva e commercializzava calcolatori) e del MIT si uniscono per realizzare Multics, un sistema operativo time-sharing. Fino ad allora, come si è già fatto notare in precedenza, il concetto esistente è quello della elaborazione batch.



Dennis Ritchie

Il progetto non andò a buon fine per motivi finanziari, ma si riuscì lo stesso a produrre versioni funzionanti di Multics. Il progetto venne chiuso nel 1969 ma un gruppo di persone, fra i quali Ritchie e Thompson continuò a lavorare producendo alla fine la prima versione di UNIX, chiamato così in contrapposizione, anche come struttura, a Multics. Laddove il primo era complesso, quest'ultimo era semplice: il kernel è il più semplice possibile includendo le funzionalità strettamente indispensabili, alle applicazioni si demanda il resto.



Ken Thompson

La prima versione fu scritta in assembly ma c'era già l'intenzione di usare un linguaggio di alto livello e così fu scritto un linguaggio chiamato B che, modifica dopo modifica e grazie agli sforzi di Ritchie e Brian Kernighan, diventa nel 1972 il C. Il kernel fu riscritto in C avviando un processo rivoluzionario: fino a quel momento i sistemi operativi erano scritti in asssembly e quindi legati strettamente alla struttura hardware, ora, con l'utilizzo di un linguaggio ad alto livello viene introdotto il concetto di portabilità e il sistema operativo può governare hardware differenti.

Thompson andò, per un certo periodo di tempo, ad insegnare a Berkeley e quando andò via si posero le basi per quella che sarebbe diventata la BSD (Berkeley Software Distribuition), una delle distribuzioni di Unix.

Nel 1984 si stacca, dalla Bell, la AT&T Computer System che comincia a commercializzare il sistema operativo UNIX.

Richard Stallman

È pure nel 1984 che Richard Stallman si dimette dal MIT e comincia a lavorare al progetto GNU. Inizialmente il lavoro al MIT era basato sulla condivisione del codice per consentire a chiunque di apportare miglioramenti al software che utilizzava. Era quella che da allora viene chiamata cultura *hacker*. Il termine, coniato all'interno del MIT, indica una persona in grado di scrivere un programma, compatto ed elegante, per risolvere un problema. Anche il nome scelto per il progetto fa riferimento ad una prassi diffusa nella cultura hacker: quella di usare nomi ricorsivi. Infatti la sigla sta per Gnu is Not Unix.

L'obiettivo, dichiarato già a cominciare dal nome scelto per il progetto, era quello di scrivere un sistema operativo che avesse le caratteristiche di potenza e versatilità di Unix, fosse in definitiva uno dei dialetti di Unix, ma non includesse codice usato in Unix e quindi proprietario. Si trattava di riscrivere da capo un nuovo sistema operativo.

Chiaramente non basta avere il kernel per poter disporre di un sistema operativo utilizzabile e, d'altra parte, anche per poter scrivere un minimo di codice occorre dotarsi degli strumenti di lavoro. Gli sforzi di Stallman e della società da lui fondata nel 1985, la Free Software Foundation, furono inizialmente quelli di scrivere gli *attrezzi del mestiere*: nascono così Emacs l'editor tuttofare che ebbe subito un grosso successo e una larga diffusione nella sua implementazione per Unix, GCC (Gnu C Compiler) un compilatore C, che diventa in seguito anche una suite di compilatori, la shell Bash, il debugger GDB.

Uno dei meriti principali attribuiti a Stallman è quello di aver ideato anche la licenza GPL (GNU Public License) con l'obiettivo di realizzare software libero ma, principalmente, di mantenerlo tale. Era la risposta ad uno dei problemi di sempre: gli sviluppatori scrivevano e diffondevano software

freeware, poi qualche produttore se ne appropriava, lo includeva in qualche suo prodotto e applicava licenze più restrittive. Con l'applicazione della licenza GPL, una parte di codice nata libera resta libera. Un software con licenza commerciale può comprendere una parte rilasciata con licenza GPL, ma le libertà ricevute assieme alla componente GPL, devono essere mantenute (nella parte soggetta a GPL o ai suoi derivati. La libertà non pervade tutto il software) e trasmesse a quelli che la utilizzeranno successivamente. È questo quello che si vuole dire quando si afferma che la GPL è una licenza di tipo *virale*.

La scrittura del kernel, uno degli obiettivi del progetto GNU, tardava a venire alla luce, ma ci pensò nel 1991 un giovane finlandese.

Linus Torvalds

Linus Torvalds frequentava ad Helsinki un corso sui Sistemi Operativi. Il punto di riferimento universitario allora era il sistema Minix, una implementazione a scopo puramente didattico, che aveva molto seguito e a cui era dedicato un newsgroup molto frequentato. Anche Torvalds resta affascinato da Minix, compra un PC e comincia col creare due processi che funzionavano in multitasking, piano piano aggiunge un file system, la gestione dell'hard disk e si ritrova con un kernel. A questo punto decide di mettere a disposizione sul server FTP dell'università il lavoro fatto, mette in condizione il kernel di lavorare con gli altri strumenti del progetto GNU, adotta la licenza GPL e fa una cosa strana che non ha precedenti: annuncia nel newsgroup di Minix la disponibilità del software e invita chiunque a collaborare, compresi quelli frustrati da esperienze negative di software non funzionante su Minix.

La risposta fu eccezionale: tantissime persone risposero da tutto il mondo e crebbero ancora quando la rete Internet si diffuse sempre di più. La prima versione ufficiale del kernel, la 1.0, ebbe la luce, con il nome GNU/Linux, nel 1994 e da allora le versioni si succedono frutto di un modo di cooperare unico e che mette bene in evidenza E. Raymond, uno dei guru del software libero:

> *Linux stravolse gran parte di quel che credevo di sapere. Per anni avevo predicato il vangelo Unix degli strumenti agili, dei prototipi immediati e della programmazione evolutiva. Ma ero anche convinto che esistesse un punto critico di complessità al di sopra del quale si rendesse necessario un approccio centralizzato e a priori. Credevo che il software più importante (sistemi operativi e strumenti davvero ingombranti come Emacs) andasse realizzato come le cattedrali, attentamente lavorato a mano da singoli geni o piccole bande di maghi che lavoravano in splendido isolamento, senza che alcuna versione beta vedesse la luce prima del momento giusto.*
>
> *Rimasi non poco sorpreso dallo stile di sviluppo proprio di Linus Torvalds: diffondere le release presto e spesso, delegare ad altri tutto il possibile, essere aperti fino alla promiscuità. Nessuna cattedrale da costruire in silenzio e reverenza. Piuttosto, la comunità Linux assomigliava a un grande e confusionario bazar, pullulante di progetti e approcci tra loro diversi. Un bazar dal quale soltanto una serie di miracoli avrebbe potuto far emergere un sistema stabile e coerente.*
>
> *Il fatto che questo stile bazar sembrasse funzionare, e anche piuttosto bene, mi colpì come uno shock. Mentre imparavo a prenderne le misure, lavoravo sodo non soltanto sui singoli progetti, ma anche cercando di comprendere come mai il mondo Linux non soltanto non cadesse preda della confusione più totale, ma al contrario andasse rafforzandosi sempre più a una velocità a malapena immaginabile per quanti costruivano cattedrali. (Eric Raymond)*

5. Il time-sharing

Time-sharing è un termine comune per indicare i sistemi multiprogrammati e multiutente. Un sistema multiutente permette a due o più terminali di accedere allo stesso calcolatore.

Come già accennato in precedenza, i sistemi batch rappresentarono un grosso passo in avanti in termini di incremento delle prestazioni di un sistema di elaborazione, ma restava un problema: se il programma in esecuzione richiede l'intervento delle unità di I/O, la CPU resta inattiva e questo avviene tanto più spesso quanto più sono le richieste. Per poter sfruttare al meglio il tempo macchina, si può cercare di sfruttare l'inattività della CPU, quando il programma è in attesa di una operazione di I/O, impegnandola per l'esecuzione di un altro programma. Il principale scopo dei sistemi multiutente in generale, e dei sistemi time-sharing in particolare, è quello di fornire buoni tempi di risposta. I sistemi time-sharing cercano di distribuire in maniera equa le risorse fra gli utenti: le elaborazioni che richiedono maggior uso della CPU vengono fatte aspettare più a lungo.

Il sistema per fornire all'utente l'illusione di avere una macchina a propria disposizione si basa sulla suddivisione del tempo macchina in piccole parti chiamate *time-slice* e sull'assegnazione, a turno, dei time-slice ai vari programmi in esecuzione. Viene assegnato un time-slice ad un programma, quando il time-slice termina, l'esecuzione del programma viene sospesa e, si assegna un time-slice ad un altro programma, e così via. Chiaramente si avrà tanto più l'illusione di avere a disposizione una macchina quanto più sarà oculata l'assegnazione dei time-slice.

La copresenza e l'esecuzione contemporanea, nel senso già trattato, di più programmi pone problemi di due tipi:

- **Problemi di sicurezza**. Ogni utente deve avere la possibilità, come si notava prima, di avere la *propria macchina a disposizione* senza che possa avvertire in alcun modo la presenza di altri utenti. Ciò vuol dire adottare meccanismi, sia hardware che software, che non consentano ad un singolo utente, per errore, di interferire con il lavoro degli altri pur usando tutte le risorse disponibili. Per esempio un utente che deposita i propri dati nell'hard disk, deve poter essere sicuro che a questi non accedano utenti non autorizzati.
- **Problemi di condivisione risorse**. L'hard disk essendo una memoria di massa che consente l'accesso diretto, può essere utilizzato da due utenti: uno lavora in una zona, l'altro in una zona diversa e il sistema può saltare da una zona ad un'altra spostando le testine di lettura-scrittura. Esistono però anche risorse hardware che, per loro natura, non si prestano alla condivisione. Si pensi, per esempio, ad una stampante: non può essere usata da due programmi in contemporanea, non si può stampare una riga di uno e una riga di un altro. In questi casi le soluzioni adottate possono essere sia di tipo hardware con i sistemi multiprocessore, cercando di parallelizzare le operazioni, rendere per esempio la stampa di un documento indipendente dall'intervento della CPU che può dedicarsi ad altri programmi, sia di tipo software generando una stampante virtuale cui il singolo programma può dirottare i propri dati in uscita.

Per il momento si tratterà dei problemi di sicurezza. In un sistema multiutente esiste, ed è il primo utente messo a disposizione dal sistema, un utente che ha accesso a tutte le risorse disponibili e a cui è demandato il compito di creare gli *account* per gli altri utenti. L'utente di cui si parla viene chiamato *amministratore di sistema*, *superutente* o anche utente *root* (nel sistema Linux). Un utente per poter usare le risorse di una macchina è necessario che abbia un account, cioè sia stato registrato, dall'utente root, come utente con determinati diritti che, a seconda delle politiche di sicurezza adottate, sono soggette a restrizioni nelle operazioni permesse: si può andare, per

esempio, dal limitare l'accesso delle directory di sistema alla sola lettura, per arrivare anche a non permettere nemmeno la lettura o a limitare la parte di hard disk che l'utente può utilizzare.

Una volta che l'account è attivo, l'utente può accedere, per mezzo della procedura di *login*, specificando il proprio *user-id* o identificativo utente, il nome con cui è registrato come utilizzatore della macchina, e la propria *password*, una parola segreta che conferma la identità dell'utente. Quando un utente finisce il suo lavoro effettua la procedura di *logout* e si scollega dal sistema. In generale lo spegnimento della macchina è un privilegio dell'utente root e, d'altra parte, un utente non può avere il potere di spegnere il sistema: potrebbero esserci altri utenti collegati. La procedura di spegnimento o *shutdown*, forza la chiusura di programmi ancora in esecuzione e lo scollegamento forzato di eventuali utenti ancora collegati. Quando si ha necessità di effettuare, per esempio per motivi di manutenzione del sistema, lo spegnimento del sistema, può essere avviato uno shutdown differito, mandandone comunicazione, per permettere agli utenti di chiudere le proprie sessioni di lavoro in maniera normale effettuando il backup dei propri dati.

5.1 Login, logout e shutdown in Linux

La procedura di login è semplice, basta digitare i dati dell'account:

```
Localhost login: tux
Password:
...
$
```

in questo caso si effettua un login come utente con identificativo tux. La digitazione della password, per motivi di sicurezza, non ha eco sullo schermo. Subito dopo il sistema, presentando il prompt, si pone in attesa di ulteriori comandi.

Per la precisione il sistema, all'avvio, mette a disposizione 7 console virtuali in ognuna delle quali è possibile effettuare il login. Alla shell grafica è dedicata la console 7, tutte le console sono accessibili mediante la combinazione di tasti *Ctrl+Alt+Fn*. Per esempio se si vuole accedere alla prima console virtuale, quella di default se non si è scelto il sistema grafico, si preme la combinazione di tasti *Ctrl+Alt+F1*. Si può passare ad un altra console anche con la combinazione *Alt+Fn*, ma questo vale se non si è in una console grafica perché, in tal caso, viene riconosciuta solo la combinazione in congiunzione con il tasto *Ctrl*.

La procedura di logout è ancora più semplice:

```
$ exit
```

dopo la digitazione del comando, che può anche essere sostituito dalla combinazione di tasti *Ctrl+d* o dal comando `logout`, il sistema ripropone il prompt per un nuovo login.

La procedura di shutdown può essere avviata in diversi modi:

```
$ sudo shutdown –h +1 “sistema in spegnimento”
Messaggio in broadcast da tux@tux-desktop
        (/dev/pts/0) alle 17.11 ...

The system is going down for halt in 1 minute!
sistema in spegnimento
```

Il comando può essere impartito solo con i diritti di super-utente: `sudo` che precede il comando stesso permette all'utente, dopo aver specificato la propria password, di acquisire temporaneamente,

giusto quanto occorre per l'esecuzione del comando, i diritti di un altro utente, nel caso specifico l'utente root. L'opzione `-h` (halt) avvia la procedura di arresto della macchina. Altra opzione utile può essere `-r` (reboot). Subito dopo viene espresso il ritardo, in questo caso un minuto. Alternativamente poteva essere specificato `now` (subito) o anche una ora espressa nella forma hh:mm (per esempio `17:30`). L'ultima parte, opzionale, specifica il messaggio da far comparire.

Esistono anche due comandi abbreviati per la procedura di shutdown che riguardano due casi particolari:

Abbreviazione	*Comando esteso*
`halt`	`shutdown -h now`
`reboot`	`shutdown -r now`

5.2 Creare account per utenti

Per accreditare un utente all'uso del sistema, si deve creare, con i diritti di root, un account per l'utente:

```
$ sudo adduser alfa
[sudo] password for tux:
Aggiunta dell'utente «alfa» ...
Aggiunta del nuovo gruppo «alfa» (1002) ...
Aggiunta del nuovo utente «alfa» (1002) con gruppo «alfa» ...
Creazione della directory home «/home/alfa» ...
Copia dei file da «/etc/skel» ...
Inserire nuova password UNIX:
Reinserire la nuova password UNIX:
passwd: password aggiornata correttamente
Modifica delle informazioni relative all'utente alfa
Inserire il nuovo valore o premere INVIO per quello predefinito
        Nome completo []: utente di prova
        Stanza n° []:
        Numero telefonico di lavoro []:
        Numero telefonico di casa []:
        Altro []:
L'informazione è corretta? [S/n]
```

in questo modo viene generato l'account per l'utente con user-id `alfa`. L'output del comando esplicita alcune opzioni relative all'utente creato fra cui: user-id (`1002`), home personale (`/home/alfa`) a cominciare dalla quale directory l'utente creerà la propria organizzazione di files (il proprio file system) come riterrà più opportuno. L'inserimento, due volte per sicurezza, della password e, se si vuole, di informazioni aggiuntive e la conferma finale, terminano il processo di creazione dell'account dell'utente.

La password dell'utente può essere modificata in qualsiasi momento:

```
$ sudo passwd alfa
Inserire nuova password UNIX:
Reinserire la nuova password UNIX:
passwd: password aggiornata correttamente
```

o soltanto il comando `passwd` se l'utente vuole cambiare la propria password. Dopo aver specificato, nel comando, l'identificativo dell'utente, il sistema domanda la nuova password. La digitazione della password, al solito per motivi di sicurezza, viene richiesta due volte e, inoltre, non viene fornito eco nello schermo. È opportuno precisare che per modificare la password di qualsiasi

utente sono necessari i diritti di root. Il singolo utente può modificare solamente la propria password.

Le opzioni di default per la creazione degli utenti sono contenute nel file `/etc/adduser.conf`:

```
# /etc/adduser.conf: `adduser' configuration.

# The DSHELL variable specifies the default login shell on your
# system.
DSHELL=/bin/bash

# The DHOME variable specifies the directory containing users' home
# directories.
DHOME=/home
...
# FIRST_UID to LAST_UID inclusive is the range of UIDs of dynamically
# allocated user accounts.
FIRST_UID=1000
LAST_UID=29999
```

Nel frammento riportato si notano: la shell da assegnare per default, la directory dove innestare le home degli utenti, gli identificativi utente validi. I parametri di default possono essere sovrascritti specificando delle opzioni nella riga del comando `adduser`. Per un elenco completo delle opzioni previste si consulti la pagina `man` del comando (comando `man adduser`).

Sempre in `/etc/adduser.conf` è settata la directory in cui `root` mette i file che si vuole siano copiati nella home di ogni nuovo utente:

```
# The SKEL variable specifies the directory containing "skeletal" user
# files; in other words, files such as a sample .profile that will be
# copied to the new user's home directory when it is created.
SKEL=/etc/skel
```

Nei sistemi Linux (tipicamente quelli non derivati Debian) dove non esiste `adduser` è possibile utilizzare il comando `useradd` e, in tal caso le opzioni di default sono contenute in `/etc/default/useradd`:

Dal punto di vista del sistema, creare l'account di un utente significa, intanto, aggiungere per l'utente un *entry* (una riga nuova dedicata alle informazioni sull'utente) in `/etc/passwd`:

```
$ less /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
...
alfa:x:1002:1002:utente di prova,,,:/home/alfa:/bin/bash
```

Ogni utente accreditato ha un entry in `/etc/passwd` formato da diversi campi, separati dal carattere `:`, ognuno riportante una informazione sull'utente stesso. Il primo campo è lo user-id. Il secondo campo (il posto della password) contiene una `x` per indicare che viene adottato un sistema di cifratura della password. Le password crittografate sono registrate in un ulteriore file. I due numeri successivi rappresentano l'UID (User ID) e il GID (Group ID), numeri identificativi dell'utente e del gruppo principale cui l'utente appartiene. Il successivo campo contiene i dati personali dell'utente inseriti in fase di creazione dell'utente così come trattato in precedenza. Gli ultimi due campi indicano la home dell'utente e la shell assegnata all'utente.

In `etc/passwd` si ritrovano alcuni utenti generati dal sistema, per esempio l'utente `root` che ha numero utente e gruppo `0`.

Le password degli utenti, in formato crittografato, sono conservate in `/etc/shadow`. Il file, anche in lettura e per motivi di sicurezza, è accessibile solo con permessi di root:

```
$ sudo less /etc/shadow
root:$1$?J0mhU6?$1/LKjqXYnd5waU63MGYwq1:12115:0:99999:7:::
bin:*:13991:0:99999:7:::
daemon:*:13991:0:99999:7:::
...
alfa:$1$jUZyGhyh$XKh95REsZis/BHmnrsmBN.:13102:0:99999:7:::
```

per alcuni utenti, il secondo campo contiene il carattere `*`: sono gli utenti che non hanno una password.

Quando deve accedere al sistema, l'utente inserisce i dati della propria login, ovvero user-id e password. Il sistema controlla se c'è un entry in `/etc/passwd` per l'utente con quella user-id e, inoltre, se applicando l'algoritmo di cifratura alla password inserita, si ottiene la stessa informazione contenuta nell'entry dell'utente in `/etc/shadow`. Se le informazioni digitate coincidono con quelle riportate nei due file citati, viene avviata la shell specificata nell'entry dell'utente in `/etc/passwd` e l'utente si ritrova nella sua home (anche questa specificata in `/etc/passwd`).

5.3 Utenti che avviano servizi

In un sistema Linux quando si installano i vari servizi (web server Apache, server MySQL, ecc...)vengono generati una serie di utenti ognuno dedicato ad un determinato servizio: per esempio l'utente `www-data` è associato al server web Apache. Si tratta di un meccanismo di sicurezza: se un server, lanciato con privilegi di root per poter accedere alle risorse, dovesse bloccarsi, un ipotetico attaccante si troverebbe con i privilegi di root e potrebbe compromettere l'intero sistema. Se, invece, il servizio è stato avviato da un utente con privilegi ridotti, il tipo di compromissione del sistema è limitato.

```
$ less /etc/passwd
...
www-data:x:33:33:www-data:/var/www:/bin/sh
...
$ sudo less /etc/shadow
...
www-data:*:13991:0:99999:7:::
...
```

L'output del primo comando evidenzia che esiste l'utente `www-data` a cui è associata la home `/var/www` (la directory dove vanno messe le pagine web che dovrà gestire il server).

L'output del secondo comando evidenzia, per mezzo della presenza del carattere `*` come secondo campo, che non è assegnata una password per l'utente. È da notare la differenza fra non assegnare una password e assegnare una password vuota. Quando si assegna una password vuota l'utente può effettuare il login senza specificare alcuna password (sicurezza del sistema compromessa). Se, al contrario, non si assegna alcuna password, l'utente non può effettuare il login (sicurezza del sistema incrementata).

5.4 Gruppi e loro gestione

Linux come sistema multiutente deve essere in grado di garantire un sistema di permessi tali da consentire l'accesso a determinate risorse solo a determinate condizioni ma deve poter consentire anche forme di collaborazione. A tal fine definisce il concetto di gruppo come insieme di utenti che hanno necessità di accedere a determinate risorse: file o anche programmi. Per esempio in un ufficio tutti gli impiegati dell'amministrazione possono avere l'esigenza di condividere la possibilità di creare e/o modificare tutti i file di una directory.

Ogni utente che ha un account nel sistema appartiene almeno ad un gruppo con identificativo uguale a quello dell'utente (gruppo primario). Il gruppo primario, cui apparterrà l'utente creato, è definito per default in `/etc/adduser.conf`:

```
# The USERGROUPS variable can be either "yes" or "no".  If "yes" each
# created user will be given their own group to use as a default.  If
# "no", each created user will be placed in the group whose gid is
# USERS_GID (see below).
USERGROUPS=yes
```

in questo caso viene creato un gruppo con lo stesso nome dell'utente (opzione `USERGROUPS=yes`).

Tutte le operazioni di gestione dei gruppi richiedono che l'utente abbia diritti di amministrare il sistema:

```
$ sudo addgroup insieme
[sudo] password for tux:
Aggiunta del gruppo «insieme» (GID 1003) ...
Fatto.
```

Il comando, dopo aver richiesto la password dell'utente in modo da attribuire i diritti di root, crea il gruppo `insieme`. Il file `/etc/group` contiene le informazioni sui gruppi definiti:

```
$ less /etc/group
root:x:0:
daemon:x:1:
...
admin:x:115:tux
...
insieme:x:1003:
```

oltre a parecchi gruppi di sistema, si nota l'esistenza del gruppo `insieme` appena creato con GID `1003` e che, al momento non contiene utenti. La presenza dell'utente `tux` nel gruppo `admin`, come verrà chiarito in un paragrafo successivo, indica il fatto che quell'utente può assumere diritti di amministrazione del sistema e, quindi dopo specifica della propria password, gestire i gruppi.

```
$ sudo adduser tux insieme
Aggiunta dell'utente «tux» al gruppo «insieme» ...
Aggiunta dell'utente tux al gruppo insieme
Fatto.
```

Il comando aggiunge l'utente `tux` al gruppo `insieme` ed, in effetti, una successiva visualizzazione di `/etc/group` mostra l'effetto del comando:

```
$ less /etc/group
...
insieme:x:1003:tux
```

in modo similare si può eliminare un utente da un gruppo:

```
$ sudo deluser tux insieme
Rimozione dell'utente «tux» dal gruppo «insieme» ...
Fatto.
```

o, anche, eliminare completamente il gruppo:

```
$ sudo delgroup insieme
Rimozione del gruppo «insieme» ...
Fatto.
```

5.5 Eliminare, sospendere account utente

Eliminare l'account di un utente comporta la cancellazione delle informazioni dell'utente registrate nel sistema:

```
$ sudo deluser alfa
Rimozione dell'utente «alfa» ...
Attenzione: il gruppo «alfa» non ha alcun membro.
Fatto.
```

Sostanzialmente vengono eliminate le informazioni sull'utente presenti nei file `/etc/passwd` e `/etc/shadow`. Come nel caso del comando di creazione di un utente, anche qui c'è un file di configurazione che determina gli effetti di `deluser`:

```
# /etc/deluser.conf: `deluser' configuration.

# Remove home directory and mail spool when user is removed
REMOVE_HOME = 0

# Remove all files on the system owned by the user to be removed
REMOVE_ALL_FILES = 0

# Backup files before removing them. This options has only an effect if
# REMOVE_HOME or REMOVE_ALL_FILES is set.
BACKUP = 0

# target directory for the backup file
BACKUP_TO = "."
```

Nell'esempio riportato la configurazione è settata in modo da non cancellare la home dell'utente. Si sarebbe potuto decidere, settando `REMOVE_HOME = 1`, di eliminare assieme all'utente anche la sua home e, in tal caso, si potrà pure decidere se effettuare prima un backup compresso dei file dell'utente ed eventualmente in quale directory conservarlo.

La home dell'utente eliminato si può cancellare con il comando:

```
$ sudo rm -R /home/alfa
```

Si tratta di una applicazione *estensiva* della remove: il parametro `-R` specifica di applicare ricorsivamente, a partire da `/home/alfa`, la cancellazione. In pratica si cancella tutto l'albero delle directory da quel punto in poi, directory inclusa. Non è superfluo ricordare la pericolosità dell'ultima operazione: se si indica il punto di inizio errato o, peggio ancora, la root directory `/`, l'esecuzione del comando comporta la cancellazione dell'intero file system.

Quando invece occorre sospendere temporaneamente l'account di un utente, poiché il sistema controlla, oltre al nome, la password, basta aggiungere un carattere `*` all'inizio della password crittografata dell'utente in `/etc/shadow`. A questo punto, poiché la password non coincide con quella stabilita dall'utente, l'utente non può effettuare il login. Per riabilitare l'utente, basterà solo togliere l'asterisco.

5.6 Utenti, superutente, cambi di identità

L'utente `root` ha accesso a tutte le risorse del sistema, può installare periferiche e programmi che saranno disponibili per tutti gli utenti, può cancellare o creare file in qualsiasi punto del file system in maniera indipendente dai permessi settati. Poiché, inoltre, Linux mette a disposizione utility per accesso a tutte le risorse per questo utente speciale, un uso improprio della modalità root può anche comportare disfunzioni del sistema e perdita di dati.

L'utente con diritti standard può operare liberamente solo all'interno della sua home. Se installa programmi, quando possibile, o modifica alcune proprietà, quelle ammesse, tutte le modifiche effettuate valgono solo per lui.

È possibile, mediante il comando `su` (Switch User), modificare temporaneamente la propria identità trasformandosi in un altro utente o anche, per determinate operazioni e temporaneamente, nell'utente `root`. Naturalmente è necessario conoscere la password dell'utente in cui ci si vuole trasformare:

```
$ whoami
uprova
$ su tux
Password:
$ whoami
tux
$ exit
exit
$ whoami
uprova
```

intanto si richiede al sistema di stampare il nome dell'utente che ha effettuato il login: il comando `whoami` chiede al sistema di stampare l'identità attuale dell'utente. Subito dopo si richiede di cambiare l'identità nell'utente `tux` assumendone tutti i diritti. Il sistema richiede la password dell'utente di cui si richiede di assumere l'identità e, se inserita correttamente, l'utente cambia. Il comando `exit`, in questo caso, permette di ritornare all'identità precedente.

In linea teorica si potrebbe assumere l'identità di root anche se si tratta di una scelta **estremamente poco consigliabile e pericolosa** (qualsiasi errore o anche bug di programma potrebbe comportare risultati disastrosi per il sistema). Il comando `su` senza specifica del nome utente permette di trasformarsi temporaneamente nell'utente `root`. Questo potrebbe essere utile se si vuole eseguire qualche compito per cui sono richiesti i privilegi di root ma non è, dal punto di vista della sicurezza, attribuire l'identità di root, per esempio, solo per eseguire singoli comandi non ammessi per i normali utenti.

Nel caso occorra dare la possibilità agli utenti di eseguire singoli comandi si può ricorrere al pacchetto `sudo` (Switch User and DO) che fornisce la possibilità di eseguire singoli programmi con i privilegi di root senza per questo cambiare l'identità dell'utente stesso.

Il sistema sudo è formato dal file di configurazione `/etc/sudoers` e dal comando `sudo`:

- il file di configurazione serve per specificare *chi* e *cosa*:

```
$ sudo less /etc/sudoers
# /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for details on how to write a sudoers file.
#

Defaults        env_reset

# Uncomment to allow members of group sudo to not need a password
# %sudo ALL=NOPASSWD: ALL

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
```

  come evidenziato anche dal commento che precede l'ultima riga, tutti gli utenti che fanno parte del gruppo `admin` (il simbolo `%` anteposto ad un nome indica appunto che si tratta di un gruppo) possono godere dei privilegi del super-utente. Se si vogliono attribuire ad un utente poteri amministrativi basta aggiungere l'utente al gruppo.

- Il comando `sudo` serve per avviare le linee di comando specificate. Per es:

```
$ su alfa
Password:
$ whoami
alfa
$ sudo less /etc/sudoers
[sudo] password for alfa:
alfa is not in the sudoers file.  This incident will be reported.
$ exit
exit
$ whoami
tux
$ less /etc/group
...
admin:x:115:tux
```

  intanto si assume l'identità dell'utente `alfa` (il buon esito dell'operazione è confermato, dopo la convalida della password dell'utente di cui si vuole assumere l'identità, dall'output del comando successivo `whoami`). Si tenta la visualizzazione del file di configurazione: il sistema chiede la password ma subito dopo comunica che l'utente `alfa` non è abilitato ad effettuare operazioni amministrative come sui evince, immediatamente, tornando indietro e visualizzando i gruppi: l'utente `alfa` non è presente nel gruppo `admin`.

5.7 Impostazioni di default per gli utenti

Nel filesystem `/etc` sono registrati file di configurazione che regolano il comportamento del sistema nella gestione degli utenti per cui è definito l'account.

```
$ less /etc/login.defs
#
# /etc/login.defs - Configuration control definitions for the login package.
#
# Three items must be defined:  MAIL_DIR, ENV_SUPATH, and ENV_PATH.
# If unspecified, some arbitrary (and possibly incorrect) value will
# be assumed.  All other items are optional - if not specified then
# the described action or option will be inhibited.
#
# Comment lines (lines beginning with "#") and blank lines are ignored.
#
...
#
# *REQUIRED*  The default PATH settings, for superuser and normal users.
#
# (they are minimal, add the rest in the shell startup files)
ENV_SUPATH      PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
ENV_PATH        PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
...
# 022 is the "historical" value in Debian for UMASK when it was used
# 027, or even 077, could be considered better for privacy
# There is no One True Answer here : each sysadmin must make up his/her
# mind.
#UMASK          022

#
# Password aging controls:
#
#       PASS_MAX_DAYS   Maximum number of days a password may be used.
#       PASS_MIN_DAYS   Minimum number of days allowed between password changes.
#       PASS_WARN_AGE   Number of days warning given before a password expires.
#
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_WARN_AGE   7

#
# Min/max values for automatic uid selection in useradd
#
UID_MIN                  1000
UID_MAX                 60000

#
# Min/max values for automatic gid selection in groupadd
#
GID_MIN                   100
GID_MAX                 60000
...
#
# Should login be allowed if we can't cd to the home directory?
# Default in no.
#
DEFAULT_HOME    yes
...
```

`/etc/login.defs` stabilisce alcuni settaggi generali riguardanti la procedura di login e concernenti, per esempio, il valore della variabile di ambiente `PATH` per gli utenti e per `root`, le caratteristiche della password: la scadenza (il valore `99999` indica che non ci sono scadenze) oltre la quale bisogna richiedere il cambiamento. Viene stabilito il numero minimo utilizzato dalla procedura di creazione degli account, per assegnare il numero all'utente o al gruppo, ed inoltre se, al login, l'utente deve essere portato nella propria home.

Le variabili di ambiente come `PATH` sono, in generale, settate dagli applicativi per *ricordare* dove trovare alcune cose che servono all'applicativo stesso per espletare il proprio compito e determinano l'ambiente in cui l'applicativo girerà. La variabile `PATH` viene impostata nella sua configurazione minima nel file `/etc/login.defs` e consente di rintracciare le directory dove sono conservati i comandi del sistema: in questo modo quando da terminale si scrive la stringa di un comando, il sistema cerca fra le directory specificate in `PATH` l'eseguibile del comando.

Le variabili di ambiente necessarie possono essere specificate, a seconda la necessità, in alcuni file che vengono eseguiti in momenti diversi:

- `/etc/profile` viene eseguito al login di ogni utente
- `/etc/bash.bashrc` viene eseguito all'avvio di una shell di ogni utente
- `/home/tux/.profile` viene eseguito al login dell'utente tux
- `/home/tux/.bashrc` viene eseguito all'avvio di una shell interattiva per l'utente tux

Nei file specificati possono essere inseriti comandi che si vuole vengano avviati nel momento specificato.

Una variabile di ambiente settata in uno dei file sopra specificati, sovrascrive il settaggio eventualmente effettuato per la stessa variabile nel `/etc/login.defs`. La riga in cui è definita la variabile `UMASK` per stabilire i diritti di default utilizzati per gli utenti, è commentata ma la variabile stessa è ridefinita in `/etc/profile`.

6. Processo e sua immagine

I sistemi multiutente multiprogrammati si basano sul modello a processi. Tutto il software eseguibile sul calcolatore, compreso il sistema operativo, è organizzato in un certo numero di processi. ***Un processo è un programma in esecuzione***. Concettualmente ogni processo ha una propria CPU virtuale ad esso dedicata, in realtà la vera CPU salta rapidamente da un processo ad un altro. Il sistema operativo deve provvedere a dare l'impressione che ogni utente abbia a propria disposizione la macchina con le proprie risorse anche se in ogni momento c'è un solo processo in esecuzione (il *processo corrente*).

Dal punto di vista dell'utente un processo è *qualcosa* di dinamico che consente l'esecuzione del programma. Dal punto di vista del kernel un processo è invece un insieme di aree di memoria e di strutture dati che registrano lo *stato* del processo stesso (*immagine* del processo).

La parte del kernel adibita alla gestione dei processi si deve occupare di:

- costruire e manipolare, in maniera corretta, le strutture di dati che costituiscono l'immagine del processo
- realizzare una politica di scheduling (distribuzione della risorsa CPU ai vari processi) in modo da

portare avanti, in modo efficiente, l'esecuzione dei vari processi garantendo tempi di risposta *accettabili* per tutti i processi

- gestire il traffico dei processi che necessitano contemporaneamente delle stesse risorse
- gestire in modo efficiente la memoria *distribuendola* ai vari processi che possono utilizzare la CPU e che sono in attesa da più tempo

L'immagine di un processo è costituita da:

- **Testo**: si tratta del codice eseguibile del programma. Il codice può essere *rientrante* quando ci sono più processi che eseguono lo stesso codice. In questi casi viene tenuta in memoria solo una copia del codice a cui accedono tutti i processi che ne hanno necessità.
- **Dati statici e dinamici**: si tratta della parte specifica di ogni singolo programma (i dati e il codice non rientrante). È compresa anche un'aria di memoria che può espandersi durante l'esecuzione del programma, che contiene i dati le cui dimensioni non possono essere conosciute a priori.
- **Argomenti, environment**: area di memoria contenente i parametri passati nella linea di comando che ha lanciato il programma e l'insieme delle variabili di ambiente.
- **User Area**: contiene le informazioni per la gestione del processo come indirizzi e dimensioni delle aree di memoria utilizzate per il processo, directory corrente, informazioni sull'utente. Il sistema operativo accede a quest'area solo per il processo corrente.
- **Process table area**: contiene tutte le informazioni che devono essere sempre disponibili anche quando il processo non è quello corrente, come stato del processo, identificatore del processo o PID (Process IDentificator) numero unico assegnato al processo che lo identifica in maniera univoca, informazioni per lo scheduling, contatore dei tempi.

6.1 Processi e immagini in Linux

La fase di inizializzazione di un sistema Linux crea il processo con PID `0`. A parte il primo, ogni processo viene generato da un processo *padre*. Ogni nuovo processo viene generato dalla duplicazione del processo padre, per mezzo della chiamata di sistema `fork()` (un servizio richiesto al kernel che genera, appunto, un nuovo processo). La `fork()` genera un nuovo entry nella tabella dei processi attivi con tutti i campi uguali al processo padre tranne, per esempio, quelli del PID e dell'identificativo del processo padre, dopo di ciò il processo padre provvede a mettere in coda, assieme agli altri, il processo *figlio*. Con la chiamata alla funzione di sistema `exec()` si permette al processo di sostituire la propria immagine con quella prodotta dal file eseguibile avviato.

Il sistema più semplice per avere le informazioni indispensabili sui processi avviati è usare `ps` (Process Status):

```
$ ps
  PID TTY          TIME CMD
 1618 pts/0    00:00:00 bash
 1711 pts/0    00:00:00 emacs
 1712 pts/0    00:00:00 ps
```

in questo esempio è messo in evidenza che ci sono 3 processi, di cui uno (quello con PID `1712`) è il processo associato all'eseguibile `ps` stesso. Le altre colonne indicano, nell'ordine, il terminale associato al processo, il tempo CPU utilizzato, la linea di comando da cui è stato avviato il

processo.

Per ogni processo avviato viene generata una sottodirectory nello pseudo-filesystem `/proc`, con il nome coincidente con PID e proprietario l'utente che ha lanciato il comando:

```
$ ls -l /proc/1618
-r--r--r--    1 tux   tux        0 feb 21 20:13 cmdline
lrwxrwxrwx    1 tux   tux        0 feb 21 20:13 cwd -> /home/tux
-r--------    1 tux   tux        0 feb 21 20:13 environ
lrwxrwxrwx    1 tux   tux        0 feb 21 20:13 exe -> /bin/bash
dr-x------    2 tux   tux        0 feb 21 20:13 fd
-r--r--r--    1 tux   tux        0 feb 21 20:13 maps
-rw-------    1 tux   tux        0 feb 21 20:13 mem
-r--r--r--    1 tux   tux        0 feb 21 20:13 mounts
lrwxrwxrwx    1 tux   tux        0 feb 21 20:13 root -> /
-r--r--r--    1 tux   tux        0 feb 21 20:13 stat
-r--r--r--    1 tux   tux        0 feb 21 20:13 statm
-r--r--r--    1 tux   tux        0 feb 21 20:13 status
```

nell'elenco visualizzato alcuni file sono collegamenti simbolici: nel caso dell'esempio, `cwd` serve per individuare la directory in cui si trovava l'utente che ha lanciato il processo. Ci si potrebbe, per esempio, spostare nella directory, prendendo il parametro da specificare di seguito al comando `cd`, da `/proc/1618/cwd`:

```
$ pwd
/proc/1618
$ cd < /proc/1618/cwd
$ pwd
/home/tux
```

nella riga con il comando `cd` viene usato l'*operatore di re-indirizzamento* dell'input (il simbolo `<`): invece di specificare direttamente il nome della directory dove ci si vuole spostare, si specifica il file da cui leggere il nome.

I vari file creati nella directory del processo, costituiscono le informazioni contenute nell'immagine del processo:

File	***Contenuto***
`cmdline`	La linea di comando che ha lanciato il processo
`cwd`	Collegamento alla directory di lavoro corrente
`environ`	Le variabili di ambiente del processo
`exe`	Collegamento all'eseguibile che ha generato il processo
`fd`	Sottodirectory con i file aperti dal processo
`maps`	La mappa della memoria e i relativi permessi di accesso
`mem`	Informazioni per permettere l'accesso alle pagine del processo (le varie parti in cui è diviso)
`mounts`	I punti di montaggio dei vari file system accessibili
`root`	Collegamento alla root
`stat`	Informazioni sullo stato del processo
`statm`	Informazioni sull'occupazione di memoria delle pagine
`status`	Informazioni presenti sia in `stat` che in `statm` ma in formato più leggibile

Tutte le informazioni contenute nei file sono visualizzabili, per esempio, utilizzando il comando `less`. Fa eccezione `environ`, che necessita, come riportato nella pagina `man` di `proc`, per essere visualizzato in forma più comprensibile, della linea di comando:

```
$ (cat /proc/1618/environ; echo) | tr “\000” “\n”
```

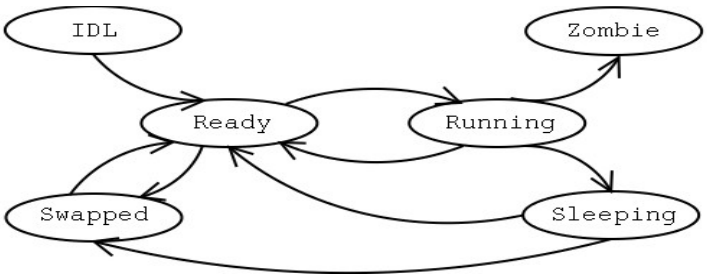
Nel file virtuale `status` sono sintetizzate le informazioni sull'immagine del processo:

```
$ less /proc/1618/status
Name:   bash
State:  S (sleeping)
Tgid:   1618
Pid:    1618
PPid:   1612
TracerPid:      0
Uid:    1003    1003    1003    1003
Gid:    100     100     100     100
FDSize: 256
Groups: 5 6 10 20 21 22 24 25 26 27 29 30 44 60 100 106 1003
VmSize:     4044 kB
VmLck:         0 kB
VmRSS:      1592 kB
VmData:      312 kB
VmStk:        20 kB
VmExe:       588 kB
VmLib:      1428 kB
...
```

dopo lo stato attuale del processo e gli identificativi del processo stesso (`Pid`), del padre (`PPid`), dell'utente, del gruppo cui appartiene l'utente, vengono elencate le dimensioni in memoria delle varie parti che compongono l'immagine.

7. Stati di un processo, transizioni di stato

Durante la sua esistenza un processo si può trovare in un determinato stato corrispondente ad una configurazione della sua immagine. Le transizioni di stato, il passaggio del processo da uno stato ad un altro, possono avvenire per effetto di chiamate al sistema operativo o in seguito al verificarsi di determinati eventi.



- **IDL (Image Definition & Loading)**. È lo stato iniziale in cui si trova un processo quando il sistema operativo prepara la sua immagine. Quando il processo è generato ed è pronto ad iniziare la sua esecuzione, viene posto allo stato READY. In questo stato non ci sono ancora risorse impegnate.
- **Ready**. Si trovano in questo stato i processi che sono in attesa di poter avere assegnata la risorsa CPU. L'ordine con cui i processi sono accodati dipende dalle politiche di scheduling. Quando il

processo corrente, quello a cui è assegnata la CPU, rilascia la risorsa, il processo che si trova per primo nella coda dei Ready, diventa il processo corrente e passa allo stato di Running. Può esserci l'esigenza di liberare spazio in memoria e, in questi casi, il processo può essere copiato in memoria di massa, passare cioè allo stato di Swapped, anche se questa probabilità è abbastanza remota e limitata a quando il sistema non riesce a liberare la memoria che serve dopo aver copiato su disco i processi bloccati.

- **Running**. In questo stato il processo utilizza la risorsa CPU. In un sistema monoprocessore ci può essere un solo processo in questo stato. Un processo in questo stato può ritornare in stato Ready quando termina il time-slice a sua disposizione e, in questo caso, il kernel porta allo stato di Ready il processo corrente e assegna la CPU al primo processo della coda Ready, che diventa ora il processo corrente. Un processo può anche decidere di sospendersi in attesa di qualche evento esterno (tipicamente operazioni di I/O) e, in questo caso, passa allo stato di Sleeping. Se invece, per mezzo di una chiamata al sistema di tipo `exit()`, il processo richiede la propria terminazione, viene posto allo stato di Zombie, in attesa di essere eliminato.
- **Sleeping**. Come già accennato prima, si trovano in questo stato i processi in attesa di un evento esterno. Questi processi sono quelli preferiti dal kernel per essere ricopiati in memoria di massa e liberare così memoria necessaria, infatti, finché l'evento richiesto non si verifica, il processo non può continuare la propria evoluzione ed è quindi inutile che occupi risorse che potrebbero essere utili a qualche altro processo che può evolversi. Quando l'evento esterno si verifica, il processo *viene risvegliato* e viene rimesso in coda allo stato di Ready.
- **Swapped**. Si trovano in questo stato i processi la cui immagine è stata copiata nella memoria di massa, nella *swap area*. Il sistema operativo può prendere in considerazione i processi in questo stato e riportarli in stato di Ready anche decidendo, per trovare lo spazio necessario, di portare in stato di Swapped qualche altro processo. Quando un processo si trova in questo stato, l'unica parte della sua immagine che è presente in memoria centrale è la process table area contenendo, questa, le informazioni necessarie alla gestione del processo stesso.
- **Zombie**. Sono i processi la cui esecuzione è terminata e che sono in attesa di essere eliminati in maniera definitiva. La loro immagine è ridotta solo alla table area e si aspetta che il processo che li ha generati raccolga il loro valore di ritorno, una specie di rapporto sull'andamento dell'esecuzione. Ricevuto il valore di ritorno, il processo padre provvede a eliminare in maniera definitiva il processo Zombie.

7.1 Job control

La pagina man della bash riporta che il job control

> *indica la capacità di fermare (sospendere) selettivamente l'esecuzione di processi e continuare (riprendere) la loro esecuzione più tardi. Tipicamente, un utente impiega questo servizio attraverso una interfaccia interattiva fornita unitamente dal driver del terminale del sistema e dalla bash.*

Per trattare questo argomento è necessario premettere alcune definizioni:

- **Job di shell**. Un programma in esecuzione genera un processo. Il programma può a sua volta mandare in esecuzione un altro programma e quindi si genera un nuovo processo. In genere con una riga di comando della shell si avvia un programma, ma può avvenire che con una singola

linea di comando si lancino più comandi, per esempio come nel caso della redirezione. Un job rappresenta tutti i processi generati da una riga di comando.

- **Processo in foreground**. In un sistema multitasking si intende con tale nome il processo corrente che può ricevere input da tastiera o usare il terminale per i suoi output.
- **Processo in background**. Si tratta di un processo che non richiede interattività con l'utente e quindi può funzionare senza impegnare il terminale.

Si può, ora, passare ad esaminare i comandi per la gestione dei job:

```
$ ps
  PID TTY          TIME CMD
 1271 tty1     00:00:00 bash
 1313 tty1     00:00:00 ps
$ emacs &
[1] 1314
$ ps u
USER    PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
tux    1271  0.0  0.4  4044 1596 pts/0    S    17:44   0:00 bash
tux    1314  0.0  2.4 12408 7948 pts/0    S    17:52   0:00 emacs
tux    1326  0.0  0.2  2572  656 pts/0    R    18:11   0:00 ps u
```

il primo comando impartito mostra i processi attualmente avviati dall'utente: il `1271` la bash e il `1313`, il comando `ps`. Nella successiva riga di comando viene avviato in background emacs. La riga di comando è terminata dal carattere `&` che serve appunto per lanciare un programma in background. Il sistema risponde con due numeri: `1` il numero di job, `1314` il PID del processo.

In questo momento non si può interagire con emacs, non si vede la schermata ma al suo posto c'è nuovamente il prompt di sistema. Il successivo comando `ps` mostra la presenza, oltre a se stesso, di due processi. In questo caso è usato il parametro `u` (User) che fornisce anche informazioni sull'utente, l'occupazione di memoria (Virtual e Resident Size) e lo stato attuale del processo. `Bash` ed `emacs` sono in stato Sleeping, `ps` è in stato Running (la colonna `STAT`).

Nel momento della visualizzazione il processo in stato di Running era quello associato all'esecuzione del comando `ps -u`. I processi associati all'esecuzione delle linee di comando `ps` e `emacs`, essendo programmi interattivi, sono in attesa di input da tastiera e quindi si trovano in stato di Sleeping.

```
$ jobs
[1]+  Running                 emacs &
```

Il comando `jobs` elenca i job avviati con il proprio numero identificativo e la riga di comando associata ad essi.

```
$ fg 1
emacs
```

il comando `fg` seguito dal numero di job, porta in primo piano emacs. Per tornare al prompt di sistema bisogna sospendere il processo attualmente in primo piano. Ciò avviene premendo la combinazione di tasti *Ctrl+z*.

```
[1]+  Stopped                 emacs
$ ps u
USER    PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
tux    1271  0.0  0.4  4044 1596 tty1     S    17:44   0:00 bash
```

```
tux    1314  0.0  2.4 12408 7948 tty1     T    17:52   0:00 emacs
tux    1325  0.0  0.2  2572  656 tty1     R    18:11   0:00 ps u
```

Il sistema risponde con il messaggio di stop: il processo si è bloccato (`STAT` riporta sTopped) ed è passato in background. Se fosse stato un programma che poteva proseguire l'esecuzione senza interazione con l'utente, si sarebbe potuto mandare un segnale di continuazione:

```
$ bg 1
[1]+ emacs &
$ ps u
USER    PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
tux    1271  0.0  0.4  4044 1596 tty1     S    17:44   0:00 bash
tux    1314  0.0  2.4 12408 7948 tty1     S    17:52   0:00 emacs
tux    1327  0.0  0.2  2572  656 tty1     R    18:11   0:00 ps u
```

il processo `emacs` è transitato in stato Sleeping.

Per terminare un processo è necessario mandare al processo un apposito segnale:

```
$ kill -SIGKILL 1314
$ ps u
USER    PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
tux    1271  0.0  0.4  4044 1596 tty1     S    17:44   0:00 bash
tux    1347  0.0  0.2  2568  652 tty1     R    18:53   0:00 ps u
[1]+  Killed                  emacs
```

il comando `kill` lancia un messaggio di terminazione e il parametro `-SIGKILL` aggiunge il fatto che la terminazione deve essere incondizionata.

7.2 Top: informazioni e gestione di processi

Il comando `top` consente la visualizzazione e il controllo sui processi:

```
$ top
top - 20:14:29 up  2:40,  2 users,  load average: 0.74, 0.98, 0.94
Tasks: 145 total,   1 running, 144 sleeping,   0 stopped,   0 zombie
Cpu(s): 12.7%us,  3.4%sy,  1.8%ni, 78.9%id,  3.1%wa,  0.1%hi,  0.2%si,  0.0%st
Mem:   2061192k total,  1553168k used,   508024k free,    66048k buffers
Swap:  4096532k total,        0k used,  4096532k free,   970124k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1797 tux       20   0  392m 123m  28m S   26  6.1  18:40.21 firefox-bin
 1006 root      20   0 58708  45m  14m S    4  2.2   8:47.61 Xorg
 1545 tux       20   0  133m  49m  22m S    2  2.5   1:55.08 cairo-dock
 1552 tux       20   0 61832  47m  14m S    2  2.4   6:54.06 compiz
 1555 tux       20   0  288m  67m  32m S    2  3.3   4:08.54 rhythmbox
 2099 tux       20   0 49792  13m  10m S    2  0.7   0:09.69 gnome-terminal
    1 root      20   0  2792 1744 1224 S    0  0.1   0:00.37 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/0
    4 root      20   0     0    0    0 S    0  0.0   0:00.04 ksoftirqd/0
    5 root      RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/1
    7 root      20   0     0    0    0 S    0  0.0   0:00.43 ksoftirqd/1
    8 root      RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/1
    ...
```

Nell'ouput del comando è evidenziata la situazione delle risorse e dell'impiego della CPU:

- la prima riga indica il nome del comando stesso, l'orario attuale, il tempo trascorso da quando il computer è in funzione, il numero di utenti connessi, il carico medio della CPU negli ultimi tempi (1 minuto, 5 minuti, 15 minuti)
- la seconda riga indica la quantità di processi suddivisi per stato
- la terza riga indica l'occupazione della CPU e le percentuali di utilizzo (`us` processi utenti, `sy` processi di sistema, `id` tempo di inattività, `wa` in attesa di I/O, `hi` ed `si` tempo dedicato ad interrupt). Informazioni utili per monitorare l'*overhead* di CPU
- le due righe successive fanno riferimento, rispettivamente, alla RAM e alla swap
- la tabella successiva mostra informazioni sui processi. Le colonne specificano oltre al PID del processo stesso e all'utente che lo ha lanciato, informazioni sulla memoria occupata (`RES` e `VIRT`), percentuale di CPU impegnata (`%CPU`) informazione utile per vedere se un processo impegna eccessivamente la risorsa. `S` è lo stato del processo (`R` running, `S` sleeping, `Z` zombie).

Top fornisce un ambiente per il controllo dei processi. Durante la sua esecuzione accetta comandi formati da un unico tasto: `h` o `?` per visualizzare una schermata con l'elenco dei comandi ammissibili, `k` per inviare il segnale di terminazione ad un processo dopo averne specificato il `PID`, `q` per terminare l'esecuzione di top.

8. Politiche di scheduling

Con questo termine si intende la strategia usata dal kernel per assegnare la risorsa CPU ai vari processi.

Un sistema di tipo time sharing deve gestire le risorse in modo tale da dare l'impressione che i processi di tutti gli utenti procedano simultaneamente. I processi si devono alternare in modo rapido, ma deve essere considerato anche che la sospensione di un processo e l'avvio di un altro richiede, da parte del sistema, un certo lavoro che non deve influire in maniera eccessiva, in calcolo percentuale, sul tempo macchina complessivo, altrimenti il sistema impiegherebbe più tempo per la sua manutenzione che per i processi utente e questo, per ovvi motivi, non può essere accettabile: ci sarebbe un eccessivo *overhead*. Può essere inoltre opportuno rendere prioritari alcuni processi di particolare importanza o che accedono a strutture molto utilizzate per evitare code di processi in attesa. Bisogna inoltre fare in modo che tutti i processi possano ottenere la CPU senza essere scavalcati continuamente da processi più importanti.

Il sistema di gestione dell'attribuzione della risorsa CPU si basa sulla *priorità di un processo* che è un numero conservato nella sua process table area. Tale priorità varia in continuazione nel corso della vita di un processo: ad un numero piccolo corrisponde un'alta priorità, a un numero più grande corrisponde una priorità più bassa. La priorità di un processo che usa molto la CPU tende a diminuire, quella di un processo che impiega molto tempo in attesa tende invece ad aumentare. Tutto ciò per non penalizzare in modo eccessivo, per esempio, i processi che eseguono molte operazioni di I/O nei confronti di quelli che ne eseguono di meno.

Un elaboratore è dotato del clock, un meccanismo hardware che genera di tanto in tanto delle interruzioni: il processo attualmente in esecuzione viene sospeso per effettuare alcune procedure di manutenzione del sistema, fra cui il calcolo della priorità dei processi. Il calcolo tiene conto di una stima dell'utilizzo della CPU da parte dei processi con un fattore correttivo (*nice*) che può essere utilizzato dall'utente per influenzare le procedure per il calcolo della priorità. Chiaramente

l'obiettivo del fattore nice non è quello di attribuire in modo esclusivo le risorse ad un solo processo, ma fare avanzare più rapidamente un processo rispetto ad un altro.

Vi sono due politiche di gestione dello scheduling che vengono eseguite periodicamente a intervalli regolari:

1. **Schedcpu**. Ricalcola la stima dell'utilizzo della CPU per tutti i processi. Se la priorità di qualche processo è diventata maggiore di quella del processo corrente, segnala la necessità di interrompere il processo corrente che avrà priorità più bassa e sarà rimpiazzato da quello con più alta priorità.
2. **Roundrobin**. Eseguita con frequenza maggiore rispetto alla prima, cerca di vedere se c'è necessità di una sostituzione del processo corrente. Tale sostituzione avviene effettivamente se c'è in coda un processo con priorità almeno uguale a quella del processo corrente.

8.1 Correzione priorità processi ed utenti

Quando sono in esecuzione più processi, il sistema cerca di distribuire il tempo macchina in modo equo fra i processi stessi. Può essere tuttavia necessario effettuare degli aggiustamenti per esempio facendo schedulare alcuni processi, che possono attendere, a quando il sistema è inattivo e lasciando più tempo a disposizione agli altri processi quando questi ne hanno necessità. La soluzione al problema è intervenire nel fattore *nice* del calcolo della priorità. In assenza di questi aggiustamenti il sistema tratta, per esempio, tutti i processi utente allo stesso modo.

Il fattore *nice*, posto al valore `0` dal sistema, può essere variato e portato in un range da `-20` (priorità massima) a `+20` (priorità minima). L'utente con diritti standard può modificare il *nice* di un fattore positivo, quindi diminuire la priorità, e solo dei processi che lui stesso ha avviato. L'utente `root` può anche aumentare la priorità di qualsiasi processo o dei processi di qualsiasi utente.

```
$ whoami
tux
$ renice -5 -u tux
renice: 1000: setpriority: Permesso negato
$ renice +1 -u tux
1000: vecchia priorità -5, nuova priorità 1
$ sudo renice -5 -u tux
1000: vecchia priorità 1, nuova priorità -5
```

Nell'esempio proposto `tux` prova ad aumentare la priorità dei propri processi ma l'operazione non viene consentita: può solo diminuirla (valore `+1`). Il comando ha effetto solo se si esegue con privilegi di `root`.

L'utente con diritti normali può lanciare un processo con priorità inferiore.

```
$ nice -10 emacs&
[1] 2239
$ nice
-5
```

Il primo comando aumenta il *nice* (il carattere `-` davanti al valore `10` indica la presenza del parametro) di un processo avviato in background e associato al comando `emacs`. Si può anche visualizzare la priorità dei propri processi con il comando `nice` senza alcun parametro.

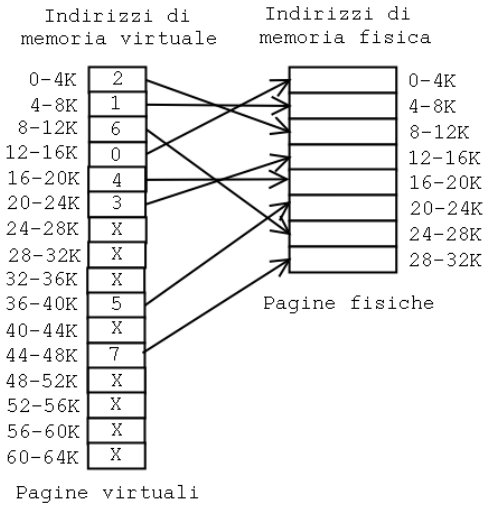
9. Gestione della memoria

La memoria centrale di un computer è una risorsa estremamente importante e delicata. Un programma per poter essere eseguito necessita di essere registrato in memoria centrale, la memoria inoltre è una risorsa limitata essendoci limiti fisici che non consentono di installare all'infinito memoria fisica in un computer. Si aggiunga a questo che la dimensione dei programmi cresce sempre di più, per esempio, per l'implementazione di interfacce utente sempre più sofisticate, ovvero come qualcuno afferma : "*i programmi si espandono per riempire la memoria che li può contenere*". Se poi si considerano i sistemi a suddivisione di tempo, si può pensare che ci siano più utenti di quanti processi possano essere caricati in memoria contemporaneamente. La parte del kernel che si occupa della gestione della risorsa memoria centrale, si chiama *memory management system*.

Per poter effettivamente avere la possibilità di gestire più utenti è necessario copiare i processi in eccesso su una memoria di massa ad accesso diretto (il disco). In questo modo si crea in memoria di massa una estensione della memoria centrale, simulando l'esistenza di molta più memoria di quanta ce ne sia effettivamente installata nel sistema. Naturalmente per far girare i processi mantenuti su disco è necessario caricarli in memoria centrale.

L'operazione di spostamento dei processi dalla memoria al disco e viceversa è chiamata ***swapping***. Nel corso degli anni sono state adottate diverse tecniche di swapping. In queste note ci si occuperà del metodo della *memoria virtuale* e, in particolare, verrà trattata la tecnica della paginazione.

L'idea di base su cui poggia la tecnica della memoria virtuale è che le dimensioni del codice e dei dati di un programma possono superare le dimensioni della memoria fisica. Se si suddivide in parti il programma e si effettua una scelta oculata delle parti da mantenere in memoria centrale si può, per esempio, far girare un programma di 1Mb in una memoria di 256Kb, suddividendo il programma in 4 pezzi da 256Kb che si alternano fra memoria centrale e disco. Se poi si è in multiprogrammazione, con lo stesso sistema usato nell'esempio precedente, in un computer con una memoria fisica di 1Mb, si possono far girare 4 programmi di 1Mb ciascuno, ognuno in un blocco di 256Kb. Ogni programma ha a disposizione una macchina privata di 256Kb di memoria centrale.



In un programma eseguibile, durante la compilazione, vengono generati un certo insieme di indirizzi di memoria che formano lo *spazio di indirizzamento virtuale*. Questo spazio viene diviso in unità dette ***pagine***. Nell'esempio si suppone che ci sia uno spazio di indirizzamento virtuale di 64K e che ogni pagina abbia dimensione 4K, ma la dimensione potrebbe essere diversa. La memoria centrale, nell'esempio con spazio di indirizzamento di 32K, viene divisa in unità della stessa dimensione delle pagine, chiamate ***frame***. La corrispondenza fra pagine virtuali e frame è chiamata *mappatura*. Per esempio la pagina 2 (8-12K) è mappata nel frame 6.

Un insieme di circuiti chiamato MMU (Memory Management Unit) si occupa di tradurre l'indirizzo

virtuale in indirizzo fisico. Il processore spedisce alla MMU l'indirizzo virtuale della locazione cui vuole effettuare un accesso, la MMU traduce l'indirizzo virtuale in indirizzo fisico e inoltra la richiesta (stavolta di un indirizzo effettivo esistente) alla memoria.

Se un programma chiede, per esempio, di accedere all'indirizzo `0`, viene mandato alla MMU l'indirizzo virtuale `0`. Poiché l'indirizzo `0` appartiene alla pagina virtuale `0` e tale pagina, secondo lo schema precedente, è mappata nel frame `2`, allora la MMU trasforma questo indirizzo in `8192` e spedisce tale indirizzo nel bus.

Con l'ausilio della MMU viene risolto il problema della corrispondenza fra indirizzi virtuali ed indirizzi fisici, ma resta un problema di fondo: in ogni caso solo 8 delle 16 pagine in cui è suddiviso il programma dell'esempio sono mappate in memoria. Se il programma richiede, per esempio, di accedere ad un indirizzo compreso nello spazio virtuale `24576-28671`, cioè compreso nella settima pagina non mappata in memoria centrale, è necessario effettuare uno swapping fra una delle pagine caricate in memoria e la pagina `7`. Se la pagina presente, in questo momento, in memoria non è stata modificata basta semplicemente che la pagina `7` venga scritta sopra la pagina rimpiazzata. Se la pagina ha subito delle modifiche bisogna conservare su disco la pagina aggiornata (potrebbe servire ancora).

Gli algoritmi per il rimpiazzamento delle pagine sono molto delicati perché devono tenere conto di parecchi fattori per non far decadere in modo significativo le prestazioni del sistema. Intanto bisogna tenere conto che ogni operazione di rimpiazzamento comporta l'uso della unità disco con un certo tempo di ritardo prima dell'esecuzione dell'operazione richiesta. Bisognerebbe quindi fare in modo da ridurre al minimo il *page swapping* evitando di scaricare pagine che possano servire immediatamente dopo.

Quando la MMU riceve la richiesta di un indirizzo virtuale appartenete ad una pagina non presente in memoria provoca una eccezione sul processore chiamata *page fault*. Il programma in esecuzione viene sospeso e viene lanciato il servizio del kernel per l'esecuzione dello swapping.

Le politiche più comunemente adottate per il page swapping sono:

- **Rimpiazzamento FIFO della pagina**. Viene rimpiazzata la pagina più vecchia nell'ipotesi supposta che, se la pagina è da tanto tempo presente in memoria, allora è probabile che ad essa non si faccia più riferimento. Questa tecnica richiede anche degli aggiustamenti che tengano conto non solo dell'età della pagina, ma anche dell'uso: una pagina potrebbe essere vecchia ma continuamente in uso.
- **Rimpiazzamento della pagina usata meno recentemente (LRU)**. Questa politica è basata sull'ipotesi che una pagina usata ultimamente verrà usata ancora in un immediato futuro. In questo caso è necessario tenere nota degli utilizzi delle pagine.

10. Gestione dei dati sulle memorie di massa

I supporti per la conservazione dei dati in formato permanente presentano problematiche di gestione differenti rispetto alla memoria centrale. Laddove l'ultima è limitata e il problema principale è quello di utilizzarla nel modo più efficiente possibile mettendo quello che serve e giusto per il tempo che serve, le memorie di massa consentono l'archiviazione di grosse masse di dati in continua modifica. Si pongono quindi problemi di velocizzazione del reperimento dei dati conservati e di organizzazione dell'enorme massa di dati che il supporto consente di mantenere.

Oggi esistono molti supporti per l'archiviazione permanente di dati dai dischi magnetici (hard disk) ai supporti ottici di vario tipo (CD, DVD). Chiaramente ogni supporto presenta delle caratteristiche peculiari e l'utilizzazione di tali supporti presupporrebbe la conoscenza di tali caratteristiche. Il sistema operativo, che ha il compito di mascherare la complessità dell'hardware e fornire una interfaccia di semplice utilizzo, mette a disposizione delle astrazioni gestite da una sua componente chiamata ***filesystem*** che ha appunto tale compito.

Il termine filesystem fino ad ora è stato usato nell'accezione riguardante l'organizzazione logica delle informazioni, ora si aggiunge una nuova definizione: si intende con questo termine la parte del kernel che si occupa della gestione delle informazioni sulle memorie di massa. A quale dei due significati ogni volta ci si riferisce, in genere, è deducibile dal contesto in cui il termine è utilizzato.

Ogni sistema operativo è in grado di gestire, in generale, più file system. Per esempio, oltre a file system che supportano caratteristiche varie come compressione, nomi di file lunghi, ci sono anche file system tipici di determinati supporti come ISO-9660 per i CD.

Dal punto di vista dell'utente esiste intanto il *file* che è semplicemente un insieme di dati contraddistinto da un nome. Un'altra astrazione messa a disposizione dell'utente è il concetto di *directory*: un contenitore che permette di raggruppare i file. Il kernel fornisce infatti un metodo per organizzare i file. Tenendo conto inoltre che i supporti di massa oggi hanno capienze molto elevate, i moderni sistemi operativi forniscono *file system di tipo gerarchico*: si possono nidificare le directory creando sotto directory nella quantità che serve, consentendo una organizzazione ordinata e razionale dei file.

10.1 Dispositivi di memorizzazione e file system

L'organizzazione dei file messa a disposizione da Linux prevede un unica organizzazione gerarchica del file system (quella che parte da `/`) cui agganciare i file system contenuti nei dispositivi di memorizzazione.

Ad ogni dispositivo di memorizzazione presente nel computer (hard disk, cd-rom, chiavette USB, partizioni dell'hard disk) Linux associa un *device*, cioè un file speciale che si trova nella `/dev`. Per esempio:

Device	***Dispositivo***
`/dev/sda`	Hard disk installato come primary master
`/dev/sdb`	Hard disk installato come primary slave
`/dev/sd..`	Altri hard disk
`/dev/cdrom`	Primo lettore cd-rom
`/dev/sda1`	Prima partizione di `sda`
`/dev/sda2`	Seconda partizione di `sda`

Il device come file permette un accesso a basso livello alla periferica associata: i dati vengono visti così come sono registrati ovvero sequenze di byte. Per poter accedere alle informazioni così come l'utente li concepisce, in termini cioè di file e sotto-directory, si associa il device ad una directory del file system. Tale operazione viene effettuata facendo il *mount* del dispositivo. Il vantaggio di questo modo di procedere è quello di avere e operare su un unico file system senza preoccuparsi dell'allocazione fisica dei files.

```
$ blkid /dev/cdrom
/dev/cdrom: LABEL="Disco dati" TYPE="iso9660"
$ sudo mount -t iso9660 /dev/cdrom /media/cdrom
mount: dispositivo a blocchi /dev/sr0 è protetto da scrittura, viene montato in
sola lettura
```

Il primo comando (BLocK IDentifier) identifica il tipo di file system di una unità di memoria. In questo caso quello presente nell'unità associata al device `/dev/cdrom`. Il secondo comando, impartito con diritti di root, aggancia il file system contenuto nel device al punto di innesto `/media/cdrom`. Da questo momento si può navigare fra il contenuto del CD accedendo alla directory `/media/cdrom`. In generale non è necessario specificare il tipo di file system (in questo caso si sarebbe potuto evitare `-t iso9660`) perché della rilevazione se ne occupa direttamente il kernel.

```
$ sudo umount /dev/cdrom
```

Il comando specificato *smonta* il dispositivo. Operazione necessaria per garantire che le modifiche che sono state effettuate sul file system (chiaramente non nel caso dei CD-ROM, ma per esempio, nel caso di un HD) siano coerenti con il contenuto del file system su memoria di massa.

In una operazione di scrittura in una memoria di massa i dati sono scritti in maniera *asincrona*: la scrittura fisica non è detto che coincida con l'operazione di scrittura che un programma o un utente compie. Per velocizzare le comunicazioni con la periferica che gestisce il supporto, il Sistema Operativo effettua una operazione fisica di scrittura sul supporto ogni volta che viene riempita una zona di memoria (*buffer*) destinata ai dati in transito per la memoria di massa. Se, per esempio, un programma di scrittura termina la sua esecuzione prima che il buffer sia completamente riempito, i dati non sono stati ancora fisicamente ricopiati sul supporto. L'operazione di unmount *forza* l'invio dei dati sincronizzandoli con la visione dell'utente.

Nel caso di CD/DVD se il dispositivo non è stato smontato il sistema impedisce l'apertura del cassetto dove è posto il supporto. Il comando `eject` apre il cassetto per espellere il supporto inserito.

L'operazione di `mount`, e del suo opposto, è in generale disponibile solo per `root`, a meno che non si sia specificato diversamente nei file di avvio del sistema.

Nel file `/proc/mounts` sono visibili i dispositivi attualmente montati con i relativi punti di innesto:

```
$ less /proc/mounts
...
/dev/hda7 /home ext3 rw,...
/dev/sr0 /media/cdrom0 iso9660 ro,...
...
```

Ogni linea specifica il device, il punto di innesto, il tipo di file system, il tipo di mounting effettuato (`rw`, indica accessibilità in lettura e scrittura, `ro` solo in lettura).

10.2 Diritti e proprietà dei file

Se si elencano i file contenuti in una directory utilizzando il parametro `l` (long) è possibile ottenere delle informazioni suppletive sui file:

```
$ ls -l
-rw-------    1 tux   tux      8434 2003-08-27 18:15 bfsh-koc.zip
-rw-r--r--    1 tux   tux     19617 2004-06-12 19:27 Blow.c
```

```
-rw-r--r--    1 tux   tux       632 2004-06-24 19:07 Blow.h
-rw-r--r--    1 tux   tux       190 2004-10-20 19:08 due.cc
drwxr-xr-x    1 tux   tux     13287 2004-10-29 19:08 duex
-rw-r--r--    1 tux   tux     17657 2005-02-01 19:42 ext2.txt
-rwxr-xr--    1 tux   tux        90 2005-02-16 19:50 filcom
```

oltre alla dimensione, alla data e ora di modifica, è riportato il proprietario del file (chi lo ha creato) e il gruppo primario cui appartiene il proprietario. Una cosa interessante è il primo gruppo di caratteri che indica i diritti sul file.

Per quanto riguarda i diritti sul file, sono previsti 10 caratteri che possono essere:

Carattere	*Significato*
`-`	Indica assenza del diritto
`d`	Specificato solo come primo carattere: indica se il file è una directory
`4` o `r`	Read: diritto di lettura sul file
`2` o `w`	Write: diritto di scrittura, modifica del file
`1` o `x`	eXecute: diritto di esecuzione

A parte il primo carattere che indica se si tratta di un file semplice o, per esempio, di una directory, il resto va letto a gruppi di 3 nella forma `rwx`: il primo gruppo da sinistra indica i diritti del proprietario, il secondo quelli del gruppo, il terzo quelli degli altri. Per esempio la stringa (divisa, per comodità di lettura, nelle sue componenti) `- rwx r-x r--` associata al file `filcom`, indica che si tratta di un file normale, che il proprietario ha tutti i diritti, che il gruppo ha diritti di lettura ed esecuzione e che gli altri hanno il solo diritto di lettura. I diritti su un file possono essere espressi anche come numeri secondo lo schema riportato nella tabella precedente, così, per esempio i permessi di `filcom` possono essere espressi con la combinazione `754` (tre numeri in ottale). `7` sono i diritti del proprietario (`r+w+x` ovvero `4+2+1`), `5` quelli del gruppo (`r+x` ovvero `4+1`), `4` quelli degli altri (`r` ovvero `4`).

I diritti assegnati per default all'utente sono visualizzabili con il comando `umask` (User MASK):

```
$ umask
0022
$ umask -S
u=rwx,g=rx,o=rx
```

nel primo caso, utilizzando il comando senza l'uso di parametro, viene fornita la maschera dei diritti: la sequenza di numeri va letta nel senso dei diritti non assegnati. Gli ultimi tre numeri indicano i diritti non assegnati rispettivamente a proprietario, gruppo, altri: il primo vale `0` cioè presenza di tutti i diritti per proprietario, i rimanenti due numeri assegnati rispettivamente al gruppo e agli altri indicano la mancanza del diritto associato al numero `2` (scrittura). Ciò in accordo anche a quanto visualizzato di conseguenza all'uso del parametro `-S` (formato simbolico). In quest'ultimo caso i diritti sono preceduti da un simbolo alfanumerico per individuare il destinatario (User, Group, Others). Tutti i file creati dall'utente avranno i diritti specificati nella maschera nel modo esposto.

Solo `root` o il proprietario del file può modificare i diritti su un file. Se si vuole, per esempio essendo valida la situazione precedente, dare, a partire dal prossimo file creato, al gruppo il diritto di scrittura si può digitare:

```
$ umask 0002
```

in questo caso, da questo momento in poi, tutti i file creati dall'utente hanno abilitato il diritto di scrittura per il gruppo.

La maschera di default per gli utenti può essere definita come variabile di ambiente `UMASK` in `/etc/login.defs` o come comando `umask` specificato in `/etc/profile`.

Per modificare i diritti solo per un singolo file:

```
$ chmod g-r due.cc
```

in questo modo si cambiano i diritti (CHange MODe) sul file `due.cc`. Viene tolto il diritto di lettura (`-r`) al gruppo (`g`). Se si voglio aggiungere diritti si usa il simbolo `+` al posto del simbolo `-`.

Si può avere la necessità di modificare il proprietario o il gruppo di un file in modo da poterlo, per esempio, condividere in modifica con tutti gli utenti di uno specifico gruppo:

```
$ sudo addgroup team
Aggiunta del gruppo «team» (GID 1002) ...
Fatto.
$ sudo adduser tux team
Aggiunta dell'utente «tux» al gruppo «team» ...
Aggiunta dell'utente tux al gruppo team
Fatto.
$ sudo adduser utente2 team
Aggiunta dell'utente «utente2» al gruppo «team» ...
Aggiunta dell'utente utente2 al gruppo team
Fatto.
$ less /etc/group
...
team:x:1002:tux,utente2
$ chown tux:team file
$ ls -l file
totale 0
-rw-r----- 1 tux team 37843 2012-01-24 17:57 file
```

Si genera il gruppo `team` e si aggiungono al gruppo gli utenti `tux` e `utente2`, dopodiché si cambiano gli accessi al file in modo che, come evidenziato dall'output dell'ultimo comando, `tux` possa modificare il file, gli utenti registrati nel gruppo `team` abbiano la possibilità di leggerne il contenuto e a tutti gli altri utenti ne sia impedito qualsiasi uso. A questo punto si potrebbe generare una directory cui possono accedere solo gli utenti del gruppo e inserire il file.

Si possono cambiare proprietario e gruppo di un file solo se si è proprietari del file stesso. Ovviamente root può modificare le proprietà di qualsiasi file di qualsiasi proprietario.

11. Gestione dello spazio su disco

I file sono registrati su dischi e quindi la gestione dello spazio disponibile è uno dei compiti del filesystem. Un file su disco può essere registrato usando due possibili metodi: allocando su disco tanto spazio contiguo quanto ne serve per contenere il file oppure suddividendo il file in tanti *blocchi* di una certa dimensione e allora, in tal caso, non è necessario che i blocchi siano contigui.

L'allocazione per byte contigui può andare bene quando i dati registrati sono statici: le dimensioni del file non si modificano con il tempo, non si cancella il file. Qualora si usasse questa tecnica, in questi ultimi due casi, bisognerebbe cancellare il file interessato alla modifica, ricompattare tutto il

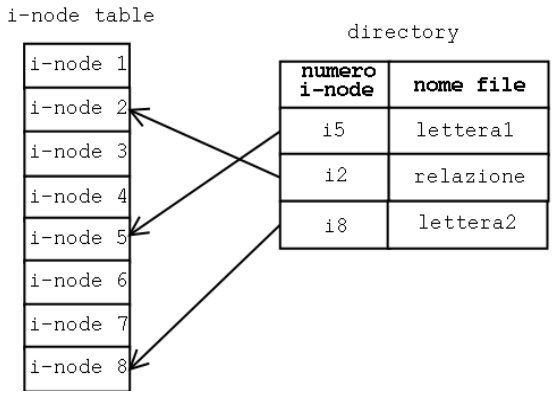
resto delle informazioni registrate nel supporto e, nel primo caso, riscrivere tutto il file alla fine. Queste sono operazioni costose in termini di prestazioni, occorre infatti un certo tempo di gestione e, quindi, questo tipo di soluzione viene adottata se i file sono scritti in un supporto e non subiscono variazioni. È quello che avviene se i file sono registrati, per esempio, in un CD.

L'allocazione per blocchi si presta meglio alla memorizzazione di file su supporti ad accesso `rw`. Fra l'altro in questo modo il cambiamento di dimensione del file può essere gestito agevolmente aggiungendo nuovi blocchi o, eventualmente, togliendone. La gestione di un file come sequenza di blocchi deve tenere in considerazione due ordini di problemi:

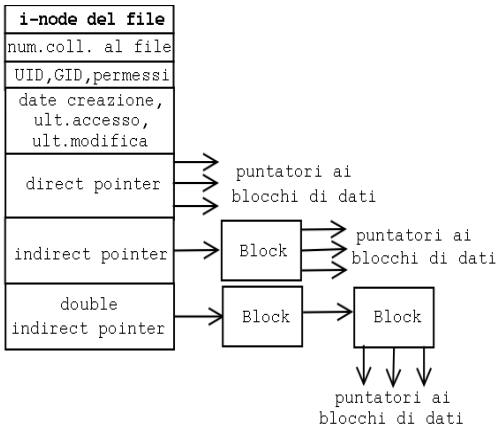
- **Dimensione del blocco**. Scegliere un blocco di dimensioni grandi può voler dire sprecare parecchio spazio. Per esempio scegliere una dimensione di 4096 byte (4K) comporta che tutti i file sono conservati in multipli di 4K: un file di 4097 byte o un file di 8000 byte occuperà sempre 2 blocchi solo che nel primo caso nel secondo blocco c'è un solo byte che interessa. D'altra parte scegliere blocchi di piccole dimensioni può comportare lentezza di reperimento delle informazioni. La lettura di un blocco da disco richiede un certo tempo per la ricerca e il posizionamento della testina di lettura/scrittura del supporto nella posizione corretta e, quindi, leggere un file composto da molti blocchi può risultare una operazione molto lenta. Nei sistemi con paginazione una soluzione può essere far coincidere la dimensione del blocco con quella della pagina.
- **Tenere traccia della sequenza di blocchi che compongono un file**. Poiché non è possibile allocare i blocchi che contengono il file uno di seguito all'altro, per gli stessi motivi trattati nell'allocazione per byte consecutivi, è necessario adottare una qualche strategia per tenere nota di tutti i blocchi in cui è spezzato un file, così come dell'insieme di blocchi ancora disponibili nel supporto. Due possibili soluzioni potrebbero essere: adottare una tabella in cui sono conservati i puntatori ai blocchi iniziali di ogni file (la sequenza dei blocchi che costituiscono il file è gestita come una struttura concatenata: ogni blocco punta al blocco successivo della struttura) o conservare, per ogni file, in una struttura, una batteria di puntatori ai blocchi.

11.1 Il filesystem di Linux: ext2

Linux può gestire diversi file system, ma quello assunto per default e che può essere considerato il file system nativo è il cosiddetto *Second Extended Filesystem*, più brevemente ***ext2***, introdotto nel 1994. Si tratta di un *block based filesystem*, termine usato per indicare il fatto che la gestione dello spazio su disco è attuata facendo uso dei cosiddetti ***Block Group*** di cui sarà esaminata la composizione.

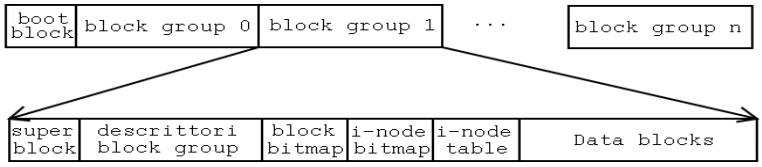


La prima struttura presa in considerazione è quella con la quale ha più contatto l'utente: la **directory**. Si tratta di un file particolare che contiene una tabella nella quale, in sintesi, sono registrati i nomi dei file e delle sottodirectory in essa contenuti. Associato ad ognuna delle voci della tabella è conservato il numero dell'*i-node* (index-node) del file. Si tratta di un puntatore ad un elemento della tabella degli i-node.



Ad ogni file è associata una struttura (l'i-node) che contiene i *metadati* del file (descrizioni dei dati del file) fra cui, per esempio, proprietario, gruppo, diritti, dimensione del file. Ci sono poi una serie di riferimenti ai blocchi fisici che contengono i dati. Detti riferimenti possono essere indiretti, cioè riferimenti a blocchi che contengono a loro volta riferimenti a blocchi fisici. In questo modo si possono referenziare anche file di grosse dimensioni.

L'allocazione dello spazio su disco avviene per blocchi e diversi blocchi vengono accorpati a formare un ***block group***. Ogni gruppo registra i file e i loro metadati in tracce del disco contigue. Ogni partizione del disco è formata da un *boot block* che registra informazioni importanti di gestione del file system e, se è il disco di boot, sul boot stesso. Seguono i block group in quantità dipendente dalle dimensioni del supporto. Il file system vero e proprio è costituito dalla sequenza dei gruppi.



Ogni singolo block group è composto da:

- **Super block**. Contiene le informazioni di base del file system come la dimensione dei blocchi, numero file, identificativo del tipo di file system (***magic number***). Il super block usato è quello registrato nel blocco 0, ma le informazioni presenti vengono ricopiate in tutti i super block per ragioni di sicurezza.
- **Descrittori di gruppi**. Anche questa è una informazione copiata in tutti i gruppi e conserva informazioni sull'identificazione di data block, data block liberi.
- **Block e i-node bitmap**. Strutture usate dal kernel per la gestione di blocchi e indici.
- **I-node table**. I metadati associati ad ogni file: quello che serve per conoscere tutte le caratteristiche ad esso associate.
- **Data blocks**. I blocchi con il contenuto vero e proprio dei file.

11.2 Evoluzione di ext2: ext3 e il journaling, ext4

Come trattato in precedenza per motivi di ottimizzazione delle prestazioni i dati non sono scritti in continuazione, ma vengono conservati in una *memoria cache* (il buffer) e, di tanto in tanto, avviene una operazione di sincronizzazione con quanto conservato su disco. In conseguenza a interruzione di energia elettrica o a crash di sistema può non essere garantita la sincronizzazione fra i dati che si erano conservati e il contenuto effettivo del file o può risultare danneggiato il file system.

Le distribuzioni Linux in genere si accorgono di una inconsistenza del file system e sono in grado di lanciare, in automatico al prossimo avvio del sistema, programmi per il ripristino del file system

danneggiato. Il problema principale è però che, all'aumentare delle dimensioni degli hard disk, le operazioni di ripristino sono sempre più gravose e richiedono tempi sempre più lunghi durante i quali, fra l'altro, la macchina non può essere utilizzata.

Per risolvere questo tipo di problema sono nati i file system di tipo journaled, uno dei quali è appunto ext3. Si tratta, in definitiva di ext2 con l'aggiunta del journaled: il kernel scrive in un apposita area del disco, chiamata journal, le modifiche effettuate. In caso di chiusure improprie del sistema, basta controllare il journal per essere in grado di riparare le incongruenze, senza andare ad effettuare il controllo di tutte le directory e i file ed, inoltre, diminuiscono i rischi di perdite di dati.

Ext4, ulteriore evoluzione, nasce per superare i limiti di dimensione dei file conservabili utilizzando ext3 e per migliorarne alcune prestazioni. Quando si richiede spazio per l'allocazione di un file ext4 effettua una allocazione multi-blocco a differenza di ext3 che alloca un blocco di 64KB per volta.

11.3 Informazioni sul file system

```
$ sudo dumpe2fs /dev/sda5
dumpe2fs 1.35 (28-Feb-2004)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          c834b4b5-6f5a-4ad8-a67a-c4474b8eac62
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal filetype needs_recovery sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              833952
Block count:              1664727
Reserved block count:     83236
Free blocks:              1109340
Free inodes:              727661
First block:              0
Block size:               4096
Fragment size:            4096
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         16352
Inode blocks per group:   511
Filesystem created:       Thu Mar 10 18:25:23 2005
Last mount time:          Thu Nov 17 09:10:08 2005
Last write time:          Thu Nov 17 09:10:08 2005
Mount count:              121
Maximum mount count:      -1
Last checked:             Thu Oct 20 20:22:11 2005
Check interval:           2592000 (1 month)
Next check after:         Sat Nov 19 19:22:11 2005
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal inode:            8
First orphan inode:       364074
Default directory hash:   tea
Directory Hash Seed:      5579caf9-afcf-4b56-8fb9-193f2c29ce0b
Journal backup:           inode blocks
```

```
...
Group 7: (Blocks 229376-262143)
  superblocco Backup a 229376, Descrittori di gruppo a 229377-229377
  Mappa dei bit di blocco a 229378 (+2), mappa dei bit inode a 229379 (+3)
  Tavola degli inode a 229380-229890 (+4)
  2931 free blocks, 12759 free inodes, 124 directories
  Blocchi liberi: 253069-255999
  Inode liberi: 118058-130816
...
```

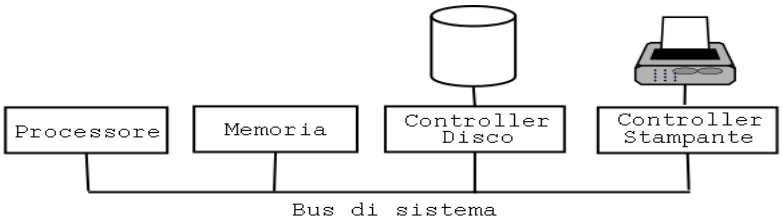
Il comando `dumpe2fs` effettua un *dumping* (scarico di dati, in questo caso, su video) del super block e dei group block, nel caso dell'esempio proposto, della partizione `/dev/sda5`. Le informazioni vanno dalla dimensione del blocco fisico (`Block size`, 4096 nel caso specifico) al numero di blocchi per gruppo (`Blocks per group`) alla quantità dei blocchi liberi (`Free blocks`). Seguono le informazioni relative a ciascun gruppo. A titolo di esempio sono riportate le informazioni sul gruppo 7 che contiene i blocchi dal 229376 al 262143. Anche qui sono riportati i blocchi liberi, la posizione della i-node table.

12. Gestione dell'I/O e dispositivi

I dispositivi di I/O collegati ad un elaboratore si possono dividere genericamente in due categorie, anche se il confine fra le due categorie non è sempre netto e non tutti i dispositivi possono essere rigidamente classificati in una o nell'altra categoria:

- **Dispositivi orientati al blocco**. Dispositivi ad accesso random. Si tratta di dispositivi che trattano le informazioni organizzate in blocchi di lunghezza fissa. Caratteristica essenziale di questi dispositivi è la possibilità di leggere o scrivere ciascun blocco in maniera indipendente dagli altri. Per esempio un disco appartiene a questa categoria: in maniera indipendente da dove si trova la testina di lettura/scrittura, l'hardware è in condizione di spostarla in una nuova posizione ed aspettare che il blocco richiesto passi sotto la testina.
- **Dispositivi orientati al carattere**. Dispositivi ad accesso sequenziale. Un dispositivo di questo tipo accetta e spedisce sequenze di caratteri senza tenere conto di alcuna struttura di blocco. Tipicamente un terminale appartiene a questa categoria, ma anche una scheda di rete, il mouse, la stampante.

Il modello dei dispositivi trattato permette al sistema operativo di procedere ad una astrazione che rende il gestore dell'input/output *indipendente dal dispositivo*. I dettagli del funzionamento del singolo dispositivo sono demandati ad un software di più basso livello: il *driver del dispositivo*



Dal punto di vista fisico le unità di I/O sono costituite da parti meccaniche ed elettroniche. La parte meccanica è il dispositivo stesso. La parte elettronica è costituita da un controllore che, spesso, assume la forma di una scheda con circuiti integrati collegata al bus di sistema. Il *controller* presenta un connettore dove si inserisce il cavo del dispositivo.

Nei grossi calcolatori si usano spesso bus multipli e processori specializzati, chiamati canali di input/output, per sgravare il processore centrale delle operazioni di trasferimento dati.

Alcuni controllori dei dispositivi orientati ai blocchi sono in grado di utilizzare il DMA (Direct Memory Access). In pratica la CPU fornisce al controllore il blocco da leggere e l'indirizzo di memoria a partire dal quale scrivere il blocco, il controllore legge il blocco, lo deposita in un suo buffer interno, controlla che tutto sia andato bene durante il trasferimento e che non ci siano stati errori, copia i dati dal buffer alla zona indicata della memoria e, alla fine, manda un segnale di interruzione. In questo modo il sistema si trova già a disposizione i dati senza essersi occupato in maniera diretta dell'incombenza delle operazioni da effettuare.

12.1 I device di Linux

In Linux, così come era anche in Unix, *ogni dispositivo è un file*. Nel file system `/dev` sono riportati i *device* ovvero i file simbolici associati ai dispositivi.

```
$ ls -l /dev/sda1
brw-rw----  1 root disk 3, 1 2001-04-15 02:43 /dev/sda1
$ ls -l /dev/tty1
crw-------  1 root root 4, 1 2005-11-17 08:10 /dev/tty1
```

Un list su alcuni device mette in evidenza il tipo di dispositivo associato ad essi; così dal primo carattere della stringa dei permessi si può vedere che la prima partizione dell'hard disk (`sda1`) è un dispositivo a blocchi come evidenzia `b` come primo carattere della stringa dei permessi e che la prima console virtuale (`tty1`) è invece un dispositivo orientato al carattere come evidenziato dal carattere `c`.

Oltre ai device associati ai dispositivi di memorizzazione già trattati in precedenza, se ne possono citare in questa sede altri che possono essere di qualche interesse

Device	*Dispositivo*
`/dev/lp0`	Line Printer. Device associato alla stampante
`/dev/zero`	Produce tanti zero quanti ne servono. Utile quando, per esempio, si vuole cancellare il contenuto di un disco
`/dev/null`	*Null device*. Tutto ciò che è mandato a questo device si perde. Una specie di *pozzo nero* dove buttare le cose. Utile, per esempio, quando non si vuole sporcare il video con i messaggi di un certo comando.

I dati conservati in un supporto gestito da una periferica possono essere resi accessibili:

- utilizzando il device che fornisce una interfaccia di *basso livello*: i dati in questo caso sono delle stringhe binarie. Utile quando si vogliono effettuare operazioni riguardanti i dati prescindendo dal loro significato logico
- montando il device in un punto del file system. In questo caso si ha la possibilità di accedere ai dati ad *alto livello*. I dati si vedono nel loro aspetto logico di aggregazione coerente: file e directory

12.2 Qualche esempio di uso dei device

- **Formattazione di un disco esterno o chiavetta USB**.

```
$ sudo mkfs.vfat /dev/sdb1
```

Nell'ipotesi che il sistema associ alla periferica il device `/dev/sdb` e che si debba formattare la prima (o unica) partizione, il comando genera un filesystem di tipo `fat` utilizzato nelle chiavette USB per ragioni di semplicità. Se servissero altri tipi di filesystem si possono utilizzare, in alternativa, i comandi `mkfs.ntfs` o `mkfs.ext2`.

- **Immagine ISO di un CD-rom**.

```
$ dd if=/dev/cdrom of=immagine.iso
766748+0 record dentro
766748+0 record fuori
392574976 byte (393 MB) copiati, 27,1499 s, 14,5 MB/s
```

Il comando `dd` (Disk Dump) effettua una copia identica: accede a basso livello al device e ne copia byte per byte il contenuto. Il parametro `if` (Input File) specifica il sorgente, il parametro `of` (Output File) la destinazione.

- **Backup di una partizione.**

```
$ sudo dd if=/dev/sda1 | gzip > partizione1.img.gz
```

In questo caso si effettua il dumping della prima partizione. L'output è filtrato (operatore |) dal comando `gzip` e inviato al file di cui viene specificato il nome. L'uso del comando `gzip` consente di avere una versione compressa della partizione altrimenti la dimensione del file risulterebbe uguale a quella della partizione e ciò potrebbe comportare maggiori difficoltà di stoccaggio.

La partizione può essere, in casi di emergenza, ripristinata dal file compresso:

```
$ sudo gzip -dc partizione1.img.gz > dd of=/dev/sda1
```

Ora, in maniera speculare rispetto alla riga precedente, è il comando `dd` a ricevere l'input dalla decompressione del file e a copiarlo nella destinazione.

Si potrebbe anche generare l'immagine dell'intero disco: basta inserire `/dev/sda` nei posti dei comandi precedenti dove è inserita la partizione. Ciò può essere utile per una installazione multipla, in un ambiente di rete con molti computer tutti con la stessa configurazione hardware. Basta configurare un solo computer e utilizzare l'immagine per trasferire la stessa configurazione negli altri computer della rete.

- **Backup della tabella delle partizioni**.

```
$ sudo dd if=/dev/sda of=mbr.out count=1 bs=512
```

In questo caso si effettua il dumping, nel file con nome `mbr.out`, dei primi 512 byte di `/dev/sda`. Essendo questo il disco di boot, i primi 512 byte sono di importanza estrema essendo quelli in cui è conservato il *Master Boot Record* ovvero il blocco di avvio del sistema oltre che la tabella delle partizioni contenente le informazioni sulle partizioni primarie.

```
$ sudo sfdisk -d /dev/sda > sda.out
```

Il comando `sfdisk` permette la manipolazione della tabella delle partizioni. In questo caso viene effettuato il dumping, sul file `sda.out`, della tabella di tutte le partizioni comprese quella estesa e quelle logiche. Il comando è utile quando esistono partizioni logiche la cui definizione non è conservata nel MBR. La conservazione delle informazioni viene effettuata in un formato

utilizzabile da `sfdisk` stesso e, quindi, l'utilizzo dei due comandi congiunti può essere una buona strategia di salvataggio da disastri. Una volta conservati su una chiavetta USB i due files generati, qualora il disco di avvio risultasse danneggiato, si potrebbe sempre rispristinare la situazione di partenza utilizzando la sequenza di comandi inversa:

```
$ sudo dd if=mbr.out of=/dev/sda
$ sudo sfdisk < sda.out
```

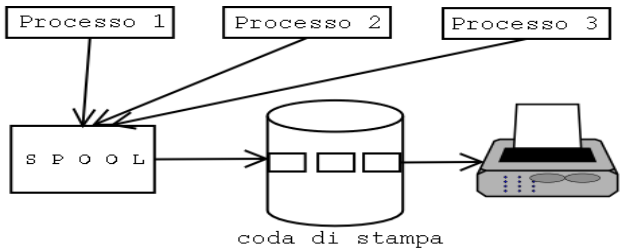
non è inutile sottolineare il significato delle ultime due operazioni descritte: si sta sovrascrivendo una parte importante del disco di avvio e si sta manipolando la tabella delle partizioni. È necessario porre la massima attenzione quando si usano comandi di tale importanza: eventuali errori commessi avrebbero, come conseguenza, effetti che comprometterebbero la funzionalità del disco.

13. Periferiche virtuali

In un sistema multiutente anche le periferiche possono generare problemi di gestione:

- **Periferiche condivise**. Si tratta di periferiche che permettono l'uso contemporaneo da parte di più utenti senza, per questo, produrre interferenze fra un processo ed un altro. Per esempio i dischi sono periferiche di questo tipo: le testine di lettura/scrittura si possono spostare da una parte ad un altra in ragione della zona di disco assegnata al processo che in quel momento ne fa richiesta. In questi casi l'uso della periferica ben si adatta ad un ambiente multiprogrammato. L'unico problema che si pone è quello di accodare le richieste in modo da ottimizzare i tempi di accesso.
- **Periferiche dedicate**. In questa categoria rientrano tutte le periferiche che debbono essere assegnate ad un processo in maniera esclusiva. Si tratta di quelle periferiche che permettono soltanto accesso sequenziale. Esempio tipico è la stampante: se un processo la sta utilizzando, la risorsa non può essere assegnata ad un altro processo, che ne fa richiesta, finché il processo che attualmente la utilizza, non la rilascia.

Le periferiche dedicate, chiaramente ferme restandone le caratteristiche, permetterebbero bassi livelli di multiutenza, limitando quest'ultima, per esempio, al numero di stampanti fisicamente collegate al calcolatore. Si consideri, inoltre, che anche se difficilmente esistono programmi che non necessitano di stampante, l'utilizzo che un programma fa di una stampante è generalmente limitato (un programma utilizza anche altre risorse) e quindi una stampante dedicata sarebbe sotto-utilizzata. L'unica soluzione possibile è quella di convertire la periferica dedicata in condivisa utilizzando le **tecniche di SPOOL** (Simultaneous Peripheral Operations On Line).



Quando si utilizzano queste tecniche, il sistema operativo, associa ad ogni processo un file su disco. Il programma di SPOOL, che gira in multiprogrammazione con gli altri, viene richiamato dalle istruzioni di scrittura di un programma in esecuzione. La scrittura sulla stampante da parte di un

programma, diventa quindi la scrittura di un file su disco. Diversi processi possono scrivere su disco essendo questo una periferica condivisa.

Quando il processo termina le operazioni di stampa, il file scritto su disco può essere materialmente stampato. La stampa avviene in modo differito e contemporaneamente ad altre esecuzioni girando, il programma di SPOOL, in multiprogrammazione con altri.

In definitiva con le tecniche di SPOOL si simula una periferica dedicata con l'utilizzo di una risorsa condivisa.

13.1 CUPS: le code di stampa in ambiente Linux

Le stampe in un sistema Linux sono gestite dal sistema CUPS (Common Unix Printing System). Si tratta di un sistema che funziona in accordo col modello client/server: si configura la stampante e si associa una coda di stampa, il server di stampa gestirà le richieste dei client.

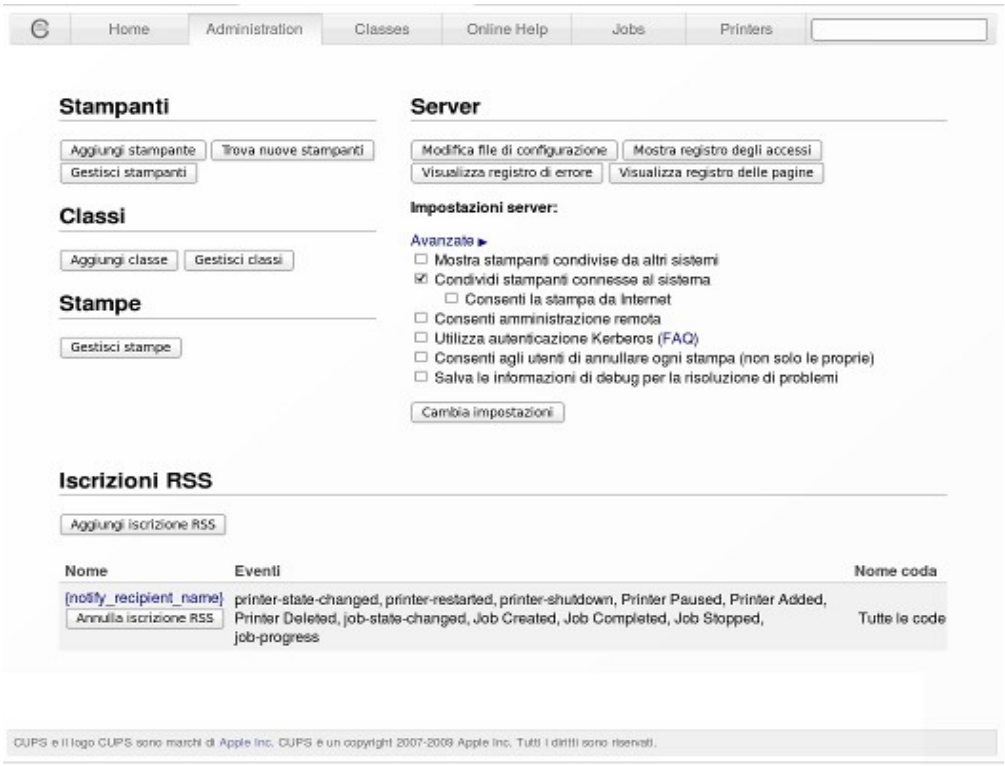
Si può interagire con CUPS per mezzo di una interfaccia web o direttamente utilizzando il prompt dei comandi.

CUPS mette a disposizione una comoda interfaccia web (collegandosi all'URL **http://localhost:631**) per l'amministrazione del server. Esistono anche browser testuali (per esempio `links2`) che non necessitano di un server grafico installato e che possono essere utilizzati per navigare fra le pagine di gestione di CUPS.

Per mezzo dell'interfaccia web si possono definire le code di stampa, gestire i job di stampa, modificare le caratteristiche delle stampanti installate. Da *Administration*, per esempio, si possono gestire le stampe o installare una nuova stampante.

Utilizzando l'interfaccia web di CUPS si può configurare facilmente il server in modo da risolvere un problema comune in un ambiente in cui esiste una rete: la condivisione di una stampante. L'ipotesi che si sta facendo riguarda una stampante collegata ad un computer della rete perché se la stampante fosse una stampante di rete il problema sarebbe già risolto: sarebbe dotata di un indirizzo IP che la identificherebbe all'interno della rete e quindi sarebbe accessibile quanto qualsiasi altro dispositivo presente nella rete stessa.

La stampante collegata ad un computer offre il vantaggio di un minore prezzo rispetto a una stampante di rete. Lo svantaggio è costituito dal fatto che è gestita dal server che gira nel computer a cui è collegata e quindi, per esempio, non è disponibile se il computer è spento.



Per condividere la stampante, dopo aver installata la stampante stessa nel computer cui è collegata, basta: dall'interfaccia di gestione scegliere il link *Administration*, selezionare *Condividi stamparti connesse al sistema* da *Impostazione server*. Negli altri computer della rete selezionare invece *Mostra stamparti condivise da altri sistemi*.

Dalla riga di comando sono disponibili una serie di comandi per la gestione delle code di stampa:

```
$ lpstat -p -d
la stampante DESKJET_840C è in attesa.  Abilitata da gio 25 set 2008 17:51:12 CEST
la stampante Generic-CUPS-PDF-Printer è in attesa.  Abilitata da ven 06 mag 2011
22:15:32 CEST
la stampante ML-2510_Series è in attesa.  Abilitata da sab 15 ott 2011 08:27:05
CEST
destinazione predefinita di sistema: Generic-CUPS-PDF-Printer
```

Innanzi tutto il comando `lpstat` (Line Printer STATus) con i parametri `-p` (printers) e `-d` (default) restituisce un elenco delle code di stampa definite e mostra la coda di destinazione di default.

```
$ cupsdisable Generic-CUPS-PDF-Printer
$ lpstat -p -d
la stampante DESKJET_840C è in attesa.  Abilitata da gio 25 set 2008 17:51:12 CEST
la stampante Generic-CUPS-PDF-Printer è disabilitata da gio 26 gen 2012 17:48:54
CET -
                                                                    Paused
la stampante ML-2510_Series è in attesa.  Abilitata da sab 15 ott 2011 08:27:05
CEST
destinazione predefinita di sistema: Generic-CUPS-PDF-Printer
```

Si disabilita la coda di stampa per evitare che la stampa dei file venga avviata immediatamente. Tutto ciò per avere la possibilità di osservare come cambia la coda di stampa e gestirla.

```
$ lpr prova1
$ lpr prova2
```

```
$ lpq
Generic-CUPS-PDF-Printer non è pronta
Posiz.  Proprietario   Stampa   Doc.             Dim. totali
1st     tux  862     prova1                        1024 byte
2nd     tux  863     prova2                        1024 byte
```

Si inviano alla coda di stampa di default due file (utilizzando il comando `lpr`) e, successivamente, si effettua una interrogazione della coda di stampa (`lpq` – Line Printer Queue). In questo caso i due job di stampa hanno codice identificativo `862` e `863`.

Se si voleva utilizzare una destinazione non di default, era necessario specificarla nella riga di comando (`lpr -P nomecoda ....`).

```
$ lprm 862
$ lprm 863
$ lpq
Generic-CUPS-PDF-Printer non è pronta
nessuna voce
```

`lprm` (Line Printer ReMove) seguito dal numero del job di stampa, elimina il job dalla coda. Una successiva interrogazione alla coda di stampa evidenzia che la coda stessa risulta vuota.

```
$ cupsenable Generic-CUPS-PDF-Printer
$ lpstat -p -d
la stampante DESKJET_840C è in attesa.  Abilitata da gio 25 set 2008 17:51:12 CEST
la stampante Generic-CUPS-PDF-Printer è in attesa.  Abilitata da gio 26 gen 2012
17:58:56 CET
la stampante ML-2510_Series è in attesa.  Abilitata da sab 15 ott 2011 08:27:05
CEST
destinazione predefinita di sistema: Generic-CUPS-PDF-Printer
```

La riabilitazione della coda di stampa (da questo momento riprende la stampa fisica dei job della coda) viene ordinata con `cupsenable`.

14. Inizializzazione del sistema

Le varie componenti del kernel hanno il compito di caricare e gestire tutti i programmi nella macchina. Normalmente il sistema operativo stesso risiede in un disco. Il problema che si pone è come fare a caricare il sistema operativo visto che è proprio il sistema operativo che si occupa del caricamento dei programmi.

La risposta al problema è data da quella che viene chiamata sequenza di lancio o ***bootstrap sequence***.

- La sequenza comincia con l'esecuzione di poche righe di codice (*bootstrap loader*) scritte permanentemente in una memoria a sola lettura e, quindi, presenti anche quando il computer è spento. L'avvio di questo programma avviene dopo che il sistema ha effettuato un test autodiagnostico sulla memoria.
- Questo piccolo programmino è in grado di accedere ai file di un particolare disco (boot disk) dove si trova il codice eseguibile del kernel del sistema.
- Caricato in memoria il kernel l'elaboratore comincia ad eseguirne il codice. Vengono inizializzate tutte le strutture dati necessarie, si inizializza il clock di sistema, vengono fatte partire le procedure di scheduling e la gestione dello swapping e a questo punto il sistema è

pronto ad iniziare ad operare in multiprogrammazione.

14.1 Inizializzazione di un sistema Linux

Il piccolo programma residente nel ROM BIOS carica da disco il settore di boot: un piccolo settore di 512 byte dove risiede un programma che legge l'immagine compressa del kernel. Questa è generalmente `/boot/vmlinuz`. A questo punto il file è scompattato in memoria e viene costruita una immagine del kernel a cui viene ceduto il controllo. Viene effettuata l'inizializzazione dei componenti hardware, montato il file system principale e lanciato `/sbin/init` che diventa il processo con ID 1, il padre di tutti i processi e che si assume il compito di lanciare i servizi necessari.

Per la scelta dei servizi, che il processo `init` avvia, Linux, tradizionalmente, utilizzava i ***runlevel***. Un runlevel è uno stato della macchina cui corrispondono alcuni servizi invece che altri. In un runlevel potrebbe essere avviato, per esempio, il web server e in un altro runlevel no. Si tratta di un numero, compreso fra 0 e 9, cui è associata una determinata configurazione:

Run level	***Modalità***
`0`	Halt
`1`	Mono utente
`2`	Multiutente ma senza supporto della rete
`3`	Multiutente
`4`	Non usata
`5`	Workstation grafica con X Window System
`6`	Reboot
`7 - 9`	Non utilizzati

Il vecchio sistema di avvio di Linux prevedeva un file di inizializzazione dei runlevel (`/etc/inittab`) e il demone *init* che avviava in sequenza i servizi. Il sistema che attualmente si avvia a sostituirlo totalmente si basa sul demone *Upstart* che avvia o ferma i servizi, in parallelo e quindi più velocemente rispetto al sistema tradizionale sequenziale, in base al verificarsi di eventi.

Il sistema si basa su:

- la directory `/etc/init` che contiene gli script di configurazione dell'avvio dei servizi

```
$ less /etc/init/rc-sysinit.conf
# rc-sysinit - System V initialisation compatibility
#
# This task runs the old System V-style system initialisation scripts,
# and enters the default runlevel when finished.

description   "System V initialisation compatibility"
author        "Scott James Remnant <scott@netsplit.com>"

start on filesystem and net-device-up IFACE=lo
stop on runlevel

# Default runlevel, this may be overriden on the kernel command-line
# or by faking an old /etc/inittab entry
env DEFAULT_RUNLEVEL=2
...
```

```
$ less /etc/init/rc.conf

# rc - System V runlevel compatibility
#
# This task runs the old System V-style rc script when changing between
# runlevels.

description   "System V runlevel compatibility"
author        "Scott James Remnant <scott@netsplit.com>"

start on runlevel [0123456]
stop on runlevel [!$RUNLEVEL]

export RUNLEVEL
export PREVLEVEL

console output
env INIT_VERBOSE

task

exec /etc/init.d/rc $RUNLEVEL
```

`rc-sysinit.conf` è il file di configurazione per `rc.conf` e stabilisce il runlevel (per i sistemi desktop con interfaccia grafica è il 2 invece del tradizionale 5). Il file rc.conf avvia `rc` con il numero di runlevel.

- La directory `/etc/default` che contiene gli script di configurazione dei servizi.

14.2 Avvio e fermo dei servizi

Nel sistema possono essere installati diversi servizi come per esempio il web server Apache, il DB server MySQL o il server di stampa. Si tratta di programmi che funzionano in background in attesa, se avviati, di richieste da parte di un programma client. In Linux sono spesso chiamati ***daemon***. I daemon sono gestiti da script di shell conservati in `/etc/init.d`:

```
$ ls -l /etc/init.d
...
-rwxr-xr-x 1 root root 6157 2010-11-18 22:16 apache2
-rwxr-xr-x 1 root root 3095 2011-09-12 16:38 cups
-rwxr-xr-x 1 root root 1329 2009-09-07 20:58 halt
lrwxrwxrwx 1 root root   21 2011-05-01 16:41 mysql -> /lib/init/upstart-job
lrwxrwxrwx 1 root root   21 2011-05-01 16:30 network-interface ->
/lib/init/upstart-job
...
```

Nell'esempio sono evidenziati alcuni script per la gestione dei server Apache, MySQL e del sistema di stampa CUPS, dei servizi di rete e di servizi utili per la chiusura del sistema (`halt`).

Ad ogni run-level corrisponde una directory del tipo `/etc/rc?.d`. Per esempio per il run level 2 la directory corrispondente è `/etc/rc2.d`. In ogni directory sono conservati tutta una serie di link simbolici ai servizi installati e che si vogliono gestire in quel run level. I collegamenti hanno nomi che cominciano con `K` (kill) o `S` (start), seguiti da un numero e dal nome del servizio. Il numero indica l'ordine con cui i vari servizi devono essere fermati o lanciati. Lo script di inizializzazione prima stoppa tutti i servizi il cui nome comincia con la `K`, nell'ordine dettato dal numero seguente, poi avvia tutti i servizi collegati ai nomi che cominciano per `S`, anche questi nell'ordine specificato

dal numero seguente. Gli script conservati nella directory relativa ad un runlevel sono avviati dopo quelli residenti in `/etc/rcS.d` che sono comuni ad ogni runlevel.

Per esempio nel caso di stop del sistema (runlevel 0):

```
$ ls -l /etc/rc0.d/
totale 0
lrwxrwxrwx 1 root root  17 2008-09-29 21:17 K09apache2 -> ../init.d/apache2
lrwxrwxrwx 1 root root  16 2008-09-24 11:31 K16dhcdbd -> ../init.d/dhcdbd
lrwxrwxrwx 1 root root  19 2008-09-29 21:43 K22mysql-ndb -> ../init.d/mysql-ndb
lrwxrwxrwx 1 root root  23 2008-09-29 21:43 K23mysql-ndb-mgm ->
../init.d/mysql-ndb-mgm
...
lrwxrwxrwx 1 root root  18 2011-05-01 16:55 S20sendsigs -> ../init.d/sendsigs
lrwxrwxrwx 1 root root  22 2008-09-24 11:31 S31umountnfs.sh ->
../init.d/umountnfs.sh
lrwxrwxrwx 1 root root  20 2011-05-01 16:30 S35networking -> ../init.d/networking
lrwxrwxrwx 1 root root  18 2008-09-24 11:31 S40umountfs -> ../init.d/umountfs
lrwxrwxrwx 1 root root  20 2008-09-24 11:31 S60umountroot -> ../init.d/umountroot
lrwxrwxrwx 1 root root  16 2009-06-14 09:39 S89casper -> ../init.d/casper
lrwxrwxrwx 1 root root  14 2008-09-24 11:31 S90halt -> ../init.d/halt
```

vengono fermati tutti i servizi avviati, dopodiché, nell'ordine, viene mandato il segnale di terminazione a eventuali programmi ancora in esecuzione, si smonta il file system e, in ultimo (`S90...`), si arresta il sistema.

Per poter rendere disponibile, in un determinato runlevel, un servizio all'avvio del sistema basta creare, nella directory relativa, un link simbolico al servizio contenuto in `/etc/init.d`, assegnando un nome composto da `S` seguito dal numero d'ordine e dal nome del servizio.

In generale, per motivi di sicurezza, è preferibile rendere disponibili sempre solo i servizi strettamente indispensabili, avviare quelli che si usano saltuariamente soltanto qualora servano e fermarli quando non servono. Gli script di gestione dei servizi prevedono, per mezzo di un parametro specificato nel lancio dello script, la possibilità di avvio e fermo. Per esempio:

```
$ sudo service apache2 start
 * Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain name,
using 127.0.1.1 for ServerName
httpd (pid 5333) already running
                                                                    [ OK ]
```

avvia il web server. Altri parametri applicabili, oltre a `start` che avvia il servizio, sono, tipicamente: `stop` per fermare il servizio, `restart` per riavviarlo, per esempio, quando si è modificato il file di configurazione e si vuole che il server rilegga la nuova configurazione. In generale, infatti, il file di configurazione viene letto all'avvio.

```
$ ps -A | grep apache
 5333 ?        00:00:00 apache2
 5336 ?        00:00:00 apache2
 5337 ?        00:00:00 apache2
 5338 ?        00:00:00 apache2
 5339 ?        00:00:00 apache2
 5340 ?        00:00:00 apache2
```

il comando `ps` seguito dal parametro `-A` premette di avere un elenco di tutti i processi attivi. Se l'output che normalmente viene inviato al monitor, viene filtrato da `grep` per la ricerca della stringa

`apache` all'interno dell'elenco, il comando restituisce informazioni sull'attività del server. Si ricorda il significato delle colonne presenti nell'output del comando: il PID del processo, il terminale associato al processo (in questo caso non c'è un terminale), il tempo CPU utilizzato, la linea di comando che ha avviato il processo.

15. Riferimenti bibliografici

Per la realizzazione di queste note sono state consultate fonti di tipo diverso:

- Intanto occorre citare le pagine `man` dei comandi che sono la fonte principale di auto-documentazione del sistema Linux
- Articoli vari pubblicati nelle riviste: Informatica Oggi, Linux PRO, Linux Pratico, Linux Magazine.
- I libri:
  - *Progettazione e Sviluppo dei Sistemi Operativi* di Andrew S. Tanenbaum. Senza dubbio un punto di riferimento per lo studio, teorico e no, dei sistemi operativi
  - *Appunti di Informatica Libera*, l'*opera omnia* di Daniele Giacomini. Testo da cui non si può assolutamente prescindere se si parla di Linux.
  - *Linux: Rute User's Tutorial and Exposition* di Paul Sheer. Un testo di tipo riassuntivo su Linux con parecchi esempi di utilizzo di comandi e alla cui lettura si deve l'idea, realizzata praticamente in queste note, di associare, alla trattazione teorica dei sistemi operativi, le applicazioni pratiche.
- La miniera di informazioni di qualsiasi genere che è *http://it.wikipedia.org*.
- La documentazione ufficiale di Ubuntu *https://help.ubuntu.com*.
- Per dovere di completezza occorre riferire anche di due articoli da cui sono tratte le due citazioni utilizzate in queste note:
  - *Il software* di Alan Kay
  - *La cattedrale e il bazar* di Eric Raymond.