Linux: guida dell'amministratore di sistema

Versione 0.6.1

Lars Wirzenius
liw@iki.fi

Introduzione all'amministrazione di un sistema Linux per principianti.

Linux: guida dell'amministratore di sistema (Versione 0.6.1)
di Lars Wirzenius

Dedica
Questa pagina è riservata ad una dedica futura.

Disponibilità del sorgente e delle versioni preformattate

Potete trovare il codice sorgente ed altri formati leggibili da computer su Internet: sono disponibili in FTP anonimo sulla home page del *Linux Documentation Project* (`http://sunsite.unc.edu/LDP/`) o sulla home page di questo libro a `http://www.iki.fi/liw/linux/sag/`, almeno in formato Postscript e TeX.DVI.

La versione italiana è disponibile in FTP anonimo dallo stesso Linux Documentation Project e sul sito del Pluto:
`ftp://ftp.pluto.linux.it/pub/pluto/ildp/`.

Indice

Elenco delle Figure **v**

Elenco delle Tabelle **vi**

1 Introduzione **1**

1.1 Il Linux Documentation Project 4

2 Panoramica di un sistema Linux **5**

2.1 Le componenti di un sistema operativo 5
2.2 Le parti principali del kernel 6
2.3 I principali servizi in un sistema UNIX 8
2.3.1 **init** 8
2.3.2 Login dai terminali 9
2.3.3 **syslog** 9
2.3.4 Esecuzione periodica dei comandi: **cron** e **at** 10
2.3.5 GUI - interfaccia grafica utente 10
2.3.6 Le reti 10
2.3.7 Login in rete 11
2.3.8 Filesystem condivisi 11
2.3.9 La posta elettronica 12
2.3.10 La stampa 12
2.3.11 La struttura del filesystem 13

3 L'albero delle directory **14**
3.1 Premesse 14
3.2 La radice del filesystem 17
3.3 La directory `/etc` 19
3.4 La directory `/dev` 21
3.5 Il filesystem `/usr` 22
3.6 Il filesystem `/var` 23
3.7 Il filesystem `/proc` 24

4 Uso dei dischi e di altre memorie di massa **27**
4.1 Due tipi di dispositivi 28
4.2 Gli hard disk 29
4.3 I floppy 33
4.4 I CD-ROM 34
4.5 Le unità nastro 36
4.6 La formattazione 36
4.7 Le partizioni 39
4.7.1 L'MBR, i settori di boot e la tabella delle partizioni . . 39
4.7.2 Partizioni estese e partizioni logiche 40
4.7.3 Tipi di partizione 41
4.7.4 Ripartizionare un hard disk 41
4.7.5 File di device e partizioni 44
4.8 I filesystem 44
4.8.1 Cosa sono i filesystem? 44
4.8.2 Filesystem a go-go 46
4.8.3 Quale filesystem bisogna usare? 48
4.8.4 Come creare un filesystem 49
4.8.5 Montare e smontare un filesystem 50
4.8.6 Il controllo dell'integrità di un filesystem con **fsck** . . . 55
4.8.7 Il controllo degli errori sul disco con **badblocks** 56
4.8.8 Combattere la frammentazione 57
4.8.9 Altri strumenti per tutti i filesystem 58
4.8.10 Altri strumenti per il filesystem ext2 58
4.9 Dischi senza filesystem 59
4.10 Allocare spazio disco 61

4.10.1 Schemi di partizionamento 61
4.10.2 Requisiti di spazio 62
4.10.3 Esempi di allocazione di hard disk 63
4.10.4 Aggiungere altro spazio disco per Linux 64
4.10.5 Suggerimenti per risparmiare spazio su disco 64

5 La gestione della memoria 65
5.1 Cosa è la memoria virtuale? 65
5.2 La creazione di uno spazio di swap 66
5.2.1 Come si usa lo spazio di swap 68
5.3 La condivisione dello spazio di swap con altri sistemi operativi 69
5.4 Allocare lo spazio di swap 69
5.5 La cache di buffer 71

6 Avvio e spengimento del sistema 74
6.1 Introduzione all'avvio ed allo spengimento del sistema 74
6.1.1 Il processo di boot più da vicino 75
6.2 Ancora sullo **shutdown** 78
6.3 Il reboot 81
6.4 Modalità utente singolo 81
6.5 Dischetti di avvio di emergenza 81

7 init 83
7.1 **init** prima di tutto 83
7.2 Configurazione di **init** per inizializzare le **getty**: `/etc/inittab` 84
7.3 I runlevel 86
7.4 Configurazioni speciali in `/etc/inittab` 87
7.5 Fare il boot in modalità utente singolo 88

8 Login e logout 90
8.1 Login via terminale 90
8.2 Login via rete 91
8.3 Cosa fa **login** 93
8.4 La shell 94
8.5 X e **xdm** 94
8.6 Il controllo degli accessi 95

9 La gestione degli account degli utenti **97**
9.1 Che cos'è un account? 97
9.2 Come creare un utente 98
9.2.1 `/etc/passwd` ed altri file informativi 98
9.2.2 Gli identificativi numerici per gli utenti e i gruppi . . . 99
9.2.3 Ambiente iniziale: `/etc/skel` 100
9.2.4 Creazione manuale di un utente 100
9.3 Modifica delle proprietà degli utenti 101
9.4 Rimozione di un utente 102
9.5 Disabilitazione temporanea di un utente 102

10 I backup **104**
10.1 Sull'importanza delle copie di backup 104
10.2 La scelta del mezzo di backup 105
10.3 La scelta dello strumento di backup 107
10.4 Backup semplici 107
10.4.1 Fare backup con **tar** 108
10.4.2 Recuperare file con **tar** 110
10.5 Backup multilivello 111
10.6 Di cosa fare backup 113
10.7 Backup compressi 114

11 Tenere il tempo **115**
11.1 I fusi orari 115
11.2 Gli orologi hardware e software 117
11.3 Impostazione e lettura dell'ora 117
11.4 Quando l'orologio si sbaglia 119

▷ ***Appendici*** **120**

A Glossario **121**

Bibliografia **123**

Elenco delle figure

2.1 Alcune delle parti più importanti del kernel di Linux 7

3.1 Parti di un albero delle directory Unix. 16

4.1 Schema di un hard disk. 31
4.2 Un esempio di partizionamento di hard disk. 42
4.3 Tre filesystem separati. 51
4.4 Sono stati montati `/home` e `/usr`. 52
4.5 Esempio di output di **dumpe2fs**. 60

8.1 Login via terminale: l'interazione di **init**, **getty**, **login** e la shell. 92

10.1 Un esempio di backup multilivello. 112

Elenco delle tabelle

4.1 Tipi di partizione (dal programma **fdisk** di Linux). 43

7.1 Numeri dei runlevel 86

10.1 Un efficace schema di backup usando molti livelli 113

Capitolo 1

Introduzione

In principio, il file era informe e vuoto, e il vuoto era sulla faccia dei bit. E le dita dell'Autore si mossero sulla faccia della tastiera. E l'Autore disse "Siano le parole!", e le parole furono.

Questo manuale, la Guida dell'amministratore di sistema di Linux, descrive appunto gli aspetti di amministrazione di sistema di Linux; è rivolto a chi non sa quasi niente di amministrazione di sistema (del tipo "ma che è?"), ma che già conosce almeno le basi dell'utilizzo comune. In questo manuale non viene spiegato come installare Linux: è compito dell'*Installation and Getting Started* [1]. Più avanti verranno date altre informazioni sui manuali di Linux disponibili.

L'amministrazione di sistema comprende tutto quello che bisogna fare per mantenere un sistema informatico in una forma utilizzabile; include cose come fare il backup dei file (e il restore, se è necessario), installare nuovi programmi, creare account per i nuovi utenti (e cancellarli quando non servono più), assicurarsi che il filesystem non sia corrotto e così via. Se un computer fosse, diciamo, una casa, l'amministrazione di sistema si chiamerebbe manutenzione, e comprenderebbe pulire, riparare le finestre rotte e altre cose del genere. L'amministrazione di sistema non si chiama manutenzione perché sarebbe troppo semplicistico[1].

La struttura di questo manuale è tale che molti dei capitoli possono essere letti indipendentemente l'uno dall'altro, quindi se vi servono delle informazioni diciamo sui backup, potete leggere solo quel capitolo. Si spera che questo renda il libro più facile da usare come riferimento ma contemporaneamente lasci la possibilità di leggerne solo una piccola parte quando serve, invece che essere costretti a studiare tutto. Comunque, questo manuale è per prima cosa un corso di amministrazione di sistemi, e un manuale di riferimento solo per una coincidenza fortuita.

Questo manuale non è inteso per essere usato completamente da solo. Gran parte dell'altra documentazione di Linux è importante per gli amministratori di sistema; dopotutto questi sono semplicemente utenti con privilegi e doveri speciali. Una risorsa molto importante sono le pagine man, che dovrebbero essere sempre consultate quando trovate un comando che non vi è familiare.

Anche se questo manuale è specifico su Linux, il principio generale seguito è che debba essere utile anche per altri sistemi operativi di tipo UNIX. Sfortunatamente, dato che c'è molta varietà tra le varie versioni di UNIX in generale, ed in particolare nell'amministrazione di sistema, c'è poca speranza di poter trattare tutte le varianti. Anche solo trattare tutte le versioni di Linux è difficile, per come si è sviluppato.

Non esiste una distribuzione di Linux ufficiale, quindi persone diverse hanno configurazioni diverse, e molti usano configurazioni personalizzate. Questo libro non è specifico su una distribuzione, anche se io uso quasi solamente la Debian GNU/Linux. Quando mi è stato possibile, ho tentato di evidenziare le differenze e di spiegare diverse alternative.

Ho provato a descrivere come funziona ciascuna cosa, piuttosto che elencare semplicemente "quattro facili lezioni" per ogni argomento. Ciò significa che il libro contiene molte informazioni che non sono indispensabili per tutti: queste parti sono però contrassegnate e possono essere saltate se usate un sistema preconfigurato. Naturalmente leggere tutto aumenterà la vostra comprensione del sistema e vi dovrebbe rendere più gradevole amministrarlo.

Come tutto lo sviluppo che riguarda Linux, il mio lavoro è stato fatto su basi volontarie: l'ho fatto perché pensavo che sarebbe stato divertente, e perché

[1] Ci sono persone che la chiamano *così*, ma solo perché non hanno mai letto questo manuale, poverini.

credevo che fosse utile. Comunque, come per tutto il lavoro di volontari, c'è un limite al tempo che ci ho potuto spendere, e anche alla mia esperienza e conoscenza. Questo significa che il manuale non è necessariamente valido come sarebbe se fosse stato scritto da un guru pagato lautamente e che abbia avuto un paio d'anni per perfezionarlo. Naturalmente credo che sia piuttosto buono, ma ritenetevi avvisati.

Un caso particolare in cui ho un po' tagliato è che non ho coperto in maniera estensiva molti argomenti già spiegati dettagliatamente in altri manuali disponibili gratuitamente, in particolare per la documentazione specifica dei programmi, come i dettagli dell'uso di **mkfs**. Io descrivo solo lo scopo del programma e quanto del suo uso è necessario per lo scopo del manuale; per altre informazioni, rimando il gentile lettore a questi altri documenti. Di solito, tutta la documentazione a cui mi riferisco fa parte della documentazione completa di Linux.

Anche se ho tentato di rendere questo manuale migliore possibile, mi piacerebbe veramente sapere da voi se avete qualche idea su come migliorarlo. Errori di linguaggio, di concetto, idee per nuovi argomenti da trattare, parti riscritte, informazioni su come si comportano le varie versioni di UNIX... mi interessa tutto. Le informazioni su come contattarmi sono disponibili via World Wide Web su `http://www.iki.fi/liw/mail-to-lasu.html`.

Molte persone mi hanno aiutato con questo libro, in maniera diretta o indiretta. Vorrei ringraziare specialmente Matt Welsh per avermi dato l'ispirazione ed aver guidato LDP, Andy Oram per avermi fatto tornare al lavoro dopo aver ricevuto tanto prezioso feedback, Olaf Kirk per avermi dimostrato che era possibile, e Adam Richter della Yggdrasil ed altri per avermi dimostrato che anche altre persone possono trovarlo interessante.

Stephen Tweedie, H. Peter Anvin, Rémy Card, Theodore T'so e Stephen Tweedie mi hanno permesso di prendere in prestito il loro lavoro (e quindi far sembrare il libro più grosso e molto più imponente): un paragone tra i filesystem **xia** ed **ext2**, l'elenco dei dispositivi ed una descrizione del filesystem **ext2**; non fanno più parte del libro. Gliene sono molto grato, e mi scuso per le versioni precedenti in cui mancava un riferimento preciso.

Oltre a questi vorrei ringraziare Mark Komarinski, per avermi mandato il suo materiale nel 1993, e tutti gli articoli sull'amministrazione di sistema del Linux Journal: sono molto utili e fonte di grande ispirazione.

Moltissime persone mi hanno mandato dei commenti utili. Il piccolo buco nero che ho per archivio non mi consente di ritrovare tutti i loro nomi, ma alcuni di questi sono, in ordine alfabetico: Paul Caprioli, Ales Cepek, Marie-France Declerfayt, Dave Dobson, Olaf Flebbe, Helmut Geyer, Larry Greenfield e suo padre, Stephen Harris, Jyrki Havia, Jim Haynes, York Lam, Timothy Andrew Lister, Jim Lynch, Michael J. Micek, Jacob Navia, Dan Poirier, Daniel Quinlan, Jouni K Seppänen, Philippe Steindl, G. B. Stotte. Le mie scuse a quelli che ho dimenticato.

1.1 Il Linux Documentation Project

Il Linux Documentation Project, o LDP, è un gruppo libero di autori, correttori di bozze e curatori che lavorano insieme per fornire una documentazione completa del sistema operativo Linux. Il coordinatore generale del progetto è Greg Hankins.

Questo manuale fa parte di una collana distribuita dall'LDP, che comprende la Guida dell'Utente (*Linux Users' Guide* [2]), la Guida dell'Amministratore di Sistema (*System Administrators' Guide* [3]), la Guida dell'Amministratore di Rete (*Network Administrators' Guide* [4]) e la Guida al Kernel (*Kernel Hackers' Guide* [5]). Questi manuali sono tutti disponibili in formato sorgente, .dvi e postscript in FTP anonimo su `metalab.unc.edu` nella directory `/pub/Linux/docs/LDP`.

Incoraggiamo chiunque abbia propensione per la scrittura o l'editoria ad aggiungersi a noi per migliorare la documentazione di Linux. Se avete accesso e-mail ad Internet, potete contattare Greg Hankins a `gregh@sunsite.unc.edu`. (N.d.T.: Per quanto riguarda la traduzione in italiano, trovate le guide LDP nella sottodirectory `translations/it` delle directory di ciascuna guida inglese. Per collaborare alla traduzione di altre guide, o per eventuali correzioni di questa, contattate me, all'email `eugenia@pluto.linux.it`.)

Capitolo 2

Panoramica di un sistema Linux

E Dio guardò quello che aveva fatto, e vide che era cosa buona.
(Genesi 1:31)

Questo capitolo dà una panoramica di un sistema Linux: per prima cosa verranno descritti i principali servizi forniti, poi, senza scendere nei dettagli, i programmi che implementano tali servizi. Lo scopo di questo capitolo è dare delle nozioni sul sistema in generale, le singole componenti sono descritte in altre parti della guida.

2.1 Le componenti di un sistema operativo

Un sistema operativo UNIX è formato da un *kernel* e da *programmi di sistema*, oltre ad *applicazioni* con cui si fa il vero lavoro. Il kernel è il cuore del sistema operativo[1]: tiene traccia dei file sul disco, avvia i programmi e li fa girare contemporaneamente, assegna la memoria ed altre risorse ai vari

[1] In effetti viene spesso in modo errato considerato il sistema operativo stesso, ma non lo è: un sistema operativo fornisce molti più servizi di un semplice kernel.

processi, scambia i pacchetti con la rete e così via. Il kernel fa pochissime cose da solo, ma fornisce gli strumenti con cui si possono costruire tutti i servizi; evita anche che chiunque acceda all'hardware direttamente, forzando tutti a utilizzare gli strumenti che fornisce: in questo modo protegge gli utenti l'uno dall'altro. Gli strumenti forniti dal kernel vengono usati attraverso *chiamate di sistema*: consultate le pagine man della sezione 2 per ulteriori informazioni su questo argomento.

I programmi di sistema usano gli strumenti forniti dal kernel per implementare i vari servizi propri di un sistema operativo. I programmi di sistema, come tutti gli altri programmi, girano 'sopra al kernel', in quella che viene chiamata *modalità utente* (*user mode*). La differenza tra i programmi di sistema e gli applicativi è lo scopo: le applicazioni sono concepite per fare delle cose utili (o per giocare, se si tratta di un gioco), mentre i programmi di sistema servono per fare funzionare il sistema stesso. Un word processor è un'applicazione, **telnet** è un programma di sistema. La differenza è comunque spesso non ben definita ed è importante solo per chi è abituato a dividere tutto in categorie.

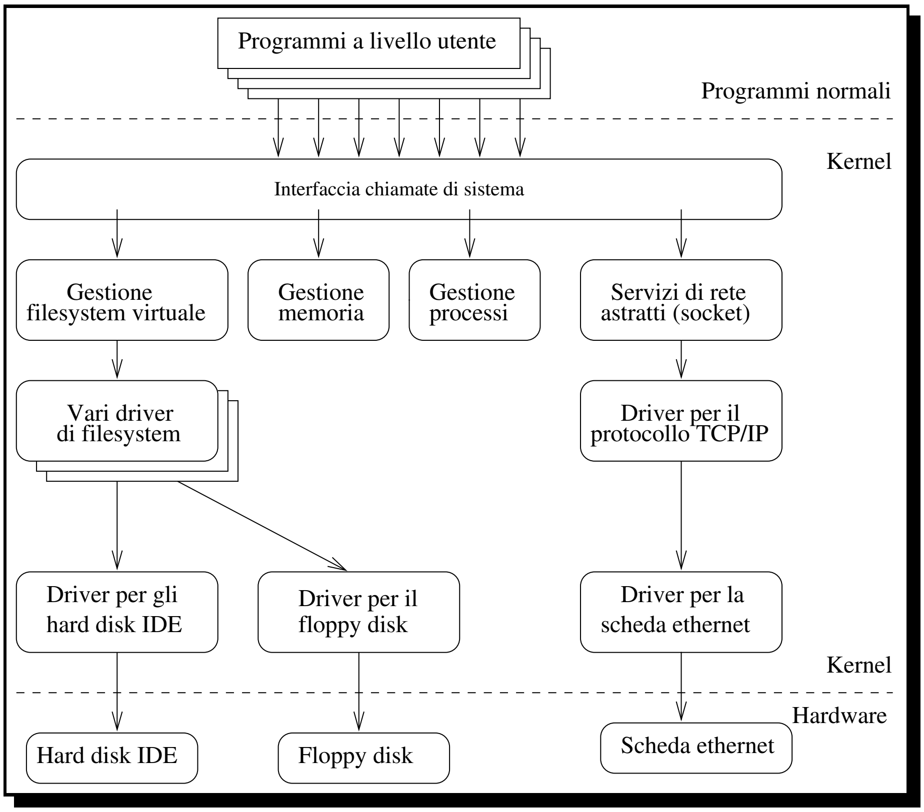
Un sistema operativo può anche contenere dei compilatori e le corrispondenti librerie (sotto Linux in particolare GCC e la libreria C), anche se non c'è bisogno che tutti i linguaggi di programmazione ne siano parte integrante. Ne possono fare parte anche la documentazione e qualche volta anche dei giochi. Tradizionalmente, il sistema operativo è stato definito come il contenuto del nastro o dei dischi di installazione; con Linux non è così chiaro, perché è sparso su tutti i siti FTP del mondo.

2.2 Le parti principali del kernel

Il kernel di Linux consiste di diverse parti importanti: la gestione dei processi, la gestione della memoria, i driver per i dispositivi hardware, quelli per i filesystem, la gestione della rete ed altre parti minori. In Figura 2.1 ne sono mostrati alcuni.

Probabilmente le parti più importanti del kernel (nient'altro funziona senza di esse) sono la gestione della memoria e dei processi. La gestione della memoria si occupa di assegnare le aree di memoria e di spazio di swap ai processi, a parti del kernel e per la cache di buffer. La gestione dei processi

Figura 2.1 – Alcune delle parti più importanti del kernel di Linux



crea i processi ed implementa il multitasking cambiando il processo attivo sul processore.

Al livello più basso, il kernel contiene un driver per ciascun dispositivo hardware che supporta. Dato che il mondo è pieno di tipi diversi di hardware il numero dei driver è enorme. Ci sono spesso molti componenti hardware simili che differiscono solo in come vengono controllati dal software; queste similitudini rendono possibile avere delle classi generali di driver che supportano operazioni simili: ogni membro della classe ha la stessa interfaccia con il resto del kernel, ma si distingue in quello che serve per implementarlo. Ad esempio, tutti i driver dei dischi sembrano uguali dal punto di vista del resto del kernel, cioè hanno tutti operazioni come 'inizializza il driver', 'leggi il settore N' e 'scrivi il settore N'.

Alcuni dei servizi software forniti dal kernel stesso hanno proprietà simili e possono dunque essere astratti in classi. Ad esempio, i vari protocolli di rete sono stati astratti in un'interfaccia di programmazione, la BSD socket library. Un altro esempio è il livello *filesystem virtuale* (VFS), che astrae le operazioni su filesystem senza considerare la loro implementazione. Ciascun tipo di filesystem fornisce un'implementazione di ogni operazione; quando un'entità prova ad usare un filesystem, la richiesta passa per il VFS che la gira al driver opportuno.

2.3 I principali servizi in un sistema UNIX

Questa sezione descrive alcuni dei servizi di UNIX più importanti, ma senza scendere molto nei dettagli; verranno descritti più accuratamente nei capitoli successivi.

2.3.1 init

Il servizio più importante di un sistema UNIX è fornito da **init**. **init** viene inizializzato come primo processo su ciascun sistema UNIX ed è l'ultima cosa che il kernel fa all'avvio. Quando parte, **init** continua il processo di avvio portando avanti vari compiti (controlla e monta i filesystem, avvia i daemon ecc.).

L'elenco esatto delle cose che **init** fa dipende dal suo tipo: ce ne sono diversi tra cui scegliere. **init** di solito fornisce il concetto di *modalità a singolo utente*, in cui nessuno si può collegare e root usa una shell alla console; la modalità usuale viene chiamata *modalità multiutente*. Alcuni tipi generalizzano questo concetto in *runlevel*; le modalità utente singolo e multiutente sono considerate due runlevel, e ce ne possono essere altri, ad esempio, per far girare X sulla console.

Nelle normali operazioni, **init** si assicura che stia girando **getty** (per poter far collegare gli utenti) e adotta i processi orfani (quelli il cui padre è morto: in UNIX *tutti* i processi devono essere in un singolo albero, quindi gli orfani devono venire adottati).

Quando il sistema viene spento, è **init** che deve uccidere tutti gli altri processi, smontare i filesystem e fermare il processore, oltre agli altri compiti per cui è stato configurato.

2.3.2 Login dai terminali

I login dai terminali (attraverso linee seriali) e dalla console (quando non si sta usando X) vengono forniti dal programma **getty**. **init** avvia una **getty** separata per ogni terminale da cui sono consentiti i login, **getty** legge il nome dell'utente ed avvia il programma **login**, che legge la password; se questa corrisponde al nome utente, **login** avvia la shell. Quando questa termina, cioè l'utente si scollega, o quando **login** termina perché il nome dell'utente e la password non corrispondono, **init** lo nota ed avvia un'altra copia di **getty**. Il kernel non ha nozione dei login, che vengono tutti gestiti dai programmi di sistema.

2.3.3 syslog

Il kernel e molti programmi di sistema producono messaggi di errore, di avvertimento e di altro tipo. Spesso è importante che questi messaggi possano essere letti in un secondo tempo, anche dopo parecchio, quindi devono essere scritti in un file. Il programma che lo fa è **syslog**, che può essere configurato per distribuire i messaggi in file diversi a seconda di chi li genera o del loro grado di importanza; ad esempio quelli del kernel sono spesso rediretti in un file separato dagli altri, dato che spesso sono più importanti e devono essere letti regolarmente per individuare gli eventuali problemi.

2.3.4 Esecuzione periodica dei comandi: cron e at

Sia gli utenti che gli amministratori di sistema hanno spesso bisogno di avviare periodicamente dei programmi, ad esempio l'amministratore di sistema potrebbe voler avviarne uno che ripulisca le directory che contengono file temporanei (`/tmp` e `/var/tmp`) dai file vecchi, per evitare che i dischi si riempiano, dato che non tutti i programmi cancellano correttamente i file generati.

Il servizio di **cron** funziona proprio per questo. Ciascun utente ha un file `crontab`, dove elenca i comandi che vuole eseguire ed il momento in cui farlo; il daemon **cron** avvia poi i comandi scelti al momento specificato.

Il servizio di **at** è simile a **cron**, ma vale per una sola volta: il comando viene eseguito al momento indicato e non viene ripetuto.

2.3.5 GUI - interfaccia grafica utente

UNIX e Linux non hanno incorporata l'interfaccia nel kernel, ma la fanno implementare da programmi a livello utente, sia per l'ambiente testuale che per quello grafico.

Questa soluzione rende il sistema più flessibile, ma ha lo svantaggio che è facile implementare un'interfaccia diversa per ciascun programma, rendendo il sistema più difficile da imparare.

L'ambiente grafico usato principalmente con Linux si chiama Sistema X Window (in breve, X). X stesso non implementa un'interfaccia utente, ma solo un sistema di finestre, cioè degli strumenti con cui poterne implementare una. I tre stili più conosciuti in X sono Athena, Motif e Open Look.

2.3.6 Le reti

Una rete è un mezzo per connettere due o più computer in modo che possano comunicare tra di loro. I metodi usati di connessione e comunicazione sono piuttosto complessi, ma il risultato finale è molto utile.

I sistemi operativi UNIX hanno molte capacità di connessione in rete. La maggior parte dei servizi di base (i filesystem, la stampa, i backup ecc.) possono essere usati attraverso la rete; questo può rendere l'amministrazione di sistema più semplice, dato che permette di centralizzare i compiti amministrativi, sfruttando nel contempo i lati positivi dei microcomputer e dell'informatica distribuita, come i costi minori e una più alta tolleranza degli errori.

Questo libro, comunque, dà solo un accenno sulle reti; per ulteriori informazioni c'è la Guida dell'amministratore di rete di Linux (*Linux Network Administrators' Guide*) [4], che comprende anche una descrizione di base del funzionamento delle reti stesse.

2.3.7 Login in rete

I login in rete funzionano in modo leggermente diverso da quelli normali: in questi ultimi infatti c'è una linea seriale fisica separata per ciascun terminale attraverso cui ci si collega, mentre per ciascuna persona che si collega via rete c'è una connessione virtuale separata, e ce ne possono essere infinite[2]. Non è quindi possibile avviare una **getty** separata per ciascuna connessione virtuale possibile. Ci sono poi diversi modi per collegarsi attraverso una rete: i principali nelle reti di tipo TCP/IP sono **telnet** e **rlogin**.

I login di rete hanno, invece di un insieme di **getty**, un daemon singolo per ciascun modo di collegamento (**telnet** e **rlogin** hanno daemon separati) che sta in ascolto per i tentativi di login in ingresso. Quando ne nota uno, inizializza una copia di se stesso per gestire quel singolo tentativo; l'istanza originale continua ad aspettarne altri. La nuova istanza funziona in modo simile a **getty**.

2.3.8 Filesystem condivisi

Una delle cose più utili che si possono fare con i servizi di rete è la condivisione di file attraverso un filesystem di rete; quello che viene usato di solito si chiama Network File System, o NFS, ed è sviluppato dalla SUN.

Con un filesystem condiviso qualsiasi operazione su file fatta da un programma su una macchina viene spedita attraverso la rete ad un altro computer. In questo modo si inganna il programma e gli si fa pensare che tutti i file sul-

[2] Beh, almeno ce ne possono essere parecchie. La larghezza di banda della rete è ancora una risorsa piuttosto scarsa, c'è ancora una limitazione pratica al numero di login contemporanei su una singola connessione di rete.

l'altro computer siano in realtà sul computer su cui sta girando, il che rende molto semplice la condivisione di informazioni, non richiedendo modifiche ai programmi stessi.

2.3.9 La posta elettronica

La posta elettronica è di solito il metodo più importante per comunicare usando il computer. Un messaggio di posta elettronica viene immagazzinato in un file in un formato speciale e per inviare e leggere i messaggi vengono usati dei programmi appositi.

Ciascun utente ha una *casella di posta in arrivo* (un file nel formato speciale) dove viene immagazzinata tutta la nuova posta in ingresso. Quando qualcuno spedisce un messaggio, il programma di posta ritrova la casella del destinatario e lo aggiunge al file. Se la casella di posta del destinatario si trova su un altro computer il messaggio viene inviato a quella macchina, che lo aggiunge alla casella nel modo che ritiene migliore.

Il sistema di posta consiste di molti programmi: la distribuzione a caselle locali o remote viene fatta dall'agente di trasferimento di posta (*mail transfer agent* o MTA), cioè **sendmail** o **smail**, mentre i programmi utilizzati dagli utenti sono molti e vari (gli agenti di posta utente, *mail user agent* o MUA, come ad esempio **pine** o **elm**). Le caselle di posta vengono in genere tenute in `/var/spool/mail`.

2.3.10 La stampa

Solo una persona alla volta può usare una stampante, ma è poco economico non condividerla tra vari utenti. La stampante viene quindi gestita da un software che implementa una *coda di stampa*: tutti i job di stampa vengono messi in una coda e quando la stampante ne ha finito uno gliene viene mandato un altro automaticamente. Questo solleva gli utenti dall'organizzazione della coda di stampa e dal litigare per il controllo della stampante[3].

Il software di stampa fa anche lo *spool* delle stampe sul disco, cioè il testo viene mantenuto in un file finché il job è in coda. Ciò permette ad un

[3]Invece formano una nuova coda *alla* stampante, aspettando le stampe, dato che nessuno sembra mai capace di far sapere esattamente al software di coda quando i vari job sono veramente finiti: un gran passo avanti per le relazioni sociali inter-ufficio.

programma applicativo di inviare velocemente i job di stampa al software di coda; l'applicazione non deve così aspettare finché il job è realmente stato stampato per poter continuare. È veramente conveniente, dato che permette di stampare una versione di un file e non dover aspettare che abbia finito per farne una nuova completamente diversa.

2.3.11 La struttura del filesystem

Il filesystem è diviso in molte parti: di solito in un filesystem radice, con `/bin`, `/lib`, `/etc` e `/dev` che formano un singolo filesystem, `/usr` con i programmi ed i dati che non vengono modificati, `/var` con quelli che vengono modificati (come i file di log) ed `/home` con i file personali dei vari utenti. A seconda della configurazione dell'hardware e delle decisioni dell'amministratore di sistema la divisione può cambiare, e potrebbe esserci anche un unico filesystem che comprende tutto.

Il Capitolo 3 descrive la struttura del filesystem abbastanza approfonditamente: il *Linux Filesystem Standard* [15] lo fa in maniera ancora più dettagliata.

Capitolo 3

L'albero delle directory

Due giorni dopo, c'era Pooh, seduto sul suo ramo, che faceva dondolare le gambe, e lì, vicino a lui, c'erano quattro vasi di miele...

(A. A. Milne)

Questo capitolo descrive le parti più importanti di un albero delle directory standard di Linux, basato sullo standard FSSTND per i filesystem; descrive come esso viene normalmente diviso in filesystem separati con scopi diversi, e spiega le motivazioni che stanno dietro questo modo di fare. Vengono descritti anche altri tipi di suddivisione.

3.1 Premesse

Questo Capitolo è basato sullo *standard dei filesystem Linux*, FSSTND, versione 1.2 [15], che tenta di impostare uno standard per l'organizzazione dell'albero delle directory nei sistemi Linux. Uno standard del genere ha il vantaggio che è più facile scrivere o fare porting del software per Linux ed amministrare le macchine Linux, dato che tutto si trova nel posto designato. Nessuna autorità impone di uniformarsi allo standard, ma questo ha il supporto della maggior parte, se non di tutte, le distribuzioni Linux, quindi non è una buona idea discostarsi da esso se non per ragioni molto particolari. Il FS-STND tenta di seguire la tradizione Unix e le tendenze più recenti, rendendo i sistemi Linux familiari per chi ha esperienza con altri sistemi Unix e viceversa.

Questo capitolo non è dettagliato come il FSSTND, e gli amministratori di sistema dovrebbero leggere anche quello per avere una comprensione completa dell'argomento; inoltre non descrive tutti i file in dettaglio: lo scopo infatti è quello di dare una visione del sistema dal punto di vista del filesystem. Ulteriori informazioni sui singoli file possono essere trovate in altre parti di questo libro o nelle pagine man.

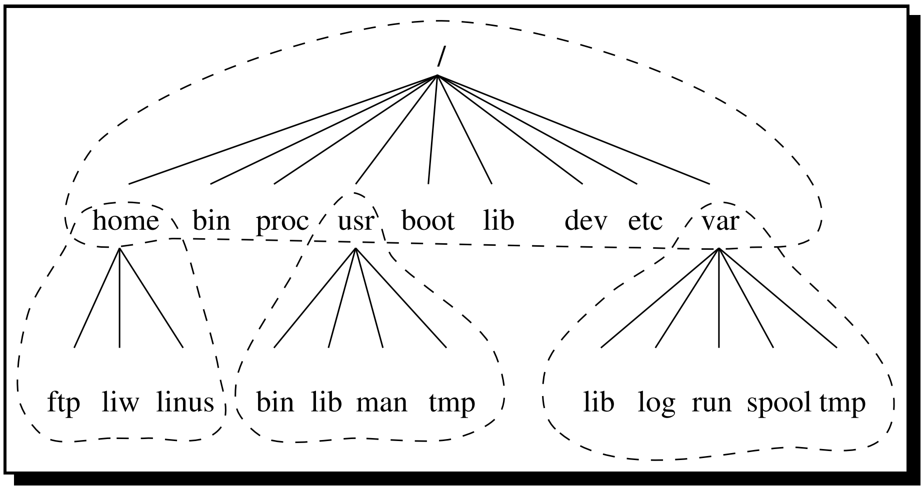
L'albero delle directory completo è concepito in modo che possa essere diviso in parti più piccole, ciascuna sulla sua partizione o nel suo disco, per ottimizzare le cose in dipendenza dalla dimensione dei dischi stessi e per rendere più semplice il backup ed il resto dell'amministrazione. Le parti principali sono i filesystem radice, `/usr`, `/var` e `/home` (vedere la Figura 3.1), e ciascuna ha uno scopo diverso. L'albero delle directory è stato strutturato in modo che funzioni bene in una rete di macchine Linux che condividano delle parti del filesystem su un dispositivo a sola lettura (ad esempio un CD-ROM), o sulla rete con NFS.

Descriviamo ora i ruoli delle diverse parti dell'albero delle directory.

- Il filesystem radice è specifico per ciascuna macchina (generalmente viene immagazzinato su un disco locale, anche se può trattarsi di un disco RAM o di uno in rete) e contiene i file necessari per avviare il sistema e per portarlo ad uno stato tale da poter montare gli altri filesystem. Il contenuto del filesystem radice sarà quindi sufficiente per la modalità a singolo utente; conterrà inoltre gli strumenti per recuperare un filesystem danneggiato o copiare dai backup i file perduti.

- Il filesystem `/usr` contiene tutti i comandi, le librerie, le pagine man e altri file che non vengono modificati durante le normali operazioni. Nessun file in `/usr` dovrebbe essere specifico per nessuna macchina data, né dovrebbe essere modificato durante il normale uso. Questo permette di condividere i file in rete, risparmiando spazio disco (`/usr` può arrivare facilmente a centinaia di megabyte) e rendendo l'amministrazione molto più semplice (basta modificare solo l'`/usr` principale quando si aggiorna un'applicazione, e non c'è bisogno di farlo separatamente su ciascuna

Figura 3.1 – *Parti di un albero delle directory Unix. Le linee tratteggiate indicano i limiti delle partizioni.*



macchina). Anche se il filesystem si trova su un disco locale può essere montato con accesso a sola lettura, per diminuire le possibilità di corruzione durante un crash.

- Il filesystem `/var` contiene file che vengono modificati, come le directory di spool (per la posta, le news, le stampanti eccetera), i file di log, le pagine man formattate ed i file temporanei. Tradizionalmente tutto quello che si trova in `/var` si trovava in qualche posto sotto `/usr`, ma questo rendeva impossibile montare `/usr` a sola lettura.

- Il filesystem `/home` contiene le home directory degli utenti, cioè tutti i veri dati sul sistema. Separare le home directory su un albero o su un filesystem a parte rende molto più semplici i backup: le altre parti in genere non ne hanno bisogno, o almeno non frequentemente (cambiano poco nel tempo). Una `/home` grande potrebbe dover essere separata in vari filesystem, aggiungendo dei sottolivelli, ad esempio `/home/students` e `/home/staff`.

Anche se le diverse parti sono state chiamate fino ad ora filesystem, non è richiesto che si trovino realmente su filesystem diversi; possono essere tenute sullo stesso se il sistema è piccolo e monoutente e si vogliono mantenere le cose semplici. L'albero delle directory può essere inoltre diviso in filesystem in modo diverso da quanto spiegato, a seconda di quanto sono grandi i dischi e di come lo spazio è allocato per i vari scopi. La parte importante, piuttosto, è che tutti i *nomi* importanti funzionino: anche se, ad esempio, `/var` e `/usr` sono in realtà sulla stessa partizione, i nomi `/usr/lib/libc.a` e `/var/adm/messages` devono funzionare; si possono ad esempio spostare i file sotto `/var` in `/usr/var`, e rendere `/var` un link simbolico a `/usr/var`.

La struttura dei filesystem Unix raggruppa i file a seconda del loro scopo, cioè tutti i comandi si trovano nello stesso posto, tutti i file di dati in un altro, la documentazione in un terzo e così via. Un'alternativa sarebbe raggruppare i file a seconda del programma a cui appartengono, cioè tutti i file di Emacs in una stessa directory, tutti quelli di $\TeX$ in un altra, eccetera. Il problema di quest'ultimo approccio è che rende difficile condividere i file (le directory dei programmi spesso contengono sia file statici che condivisibili, e sia file che cambiano che file che restano invariati) e spesso anche trovare dei file (ad esempio, le pagine man sarebbero in un numero enorme di posti e fare in modo che i programmi di lettura le possano trovare è l'incubo degli amministratori di sistema).

3.2 La radice del filesystem

Il filesystem radice dovrebbe generalmente essere piccolo, dato che contiene file estremamente critici e un filesystem piccolo che viene modificato poco ha migliori possibilità di non venire corrotto. Un filesystem radice corrotto in genere significa che diventa impossibile avviare il sistema tranne che con misure eccezionali (ad esempio da un floppy), quindi è meglio non rischiare.

La directory principale in genere non contiene nessun file, tranne a volte l'immagine standard di avvio per il sistema, che di solito si chiama `/vmlinuz`. Tutti gli altri file sono in sue sottodirectory:

`/bin`
: Comandi necessari durante l'avvio del sistema che devono essere usati dagli utenti normali (in genere dopo l'avvio).

`/sbin`
: Come `/bin`, ma i comandi non sono intesi per gli utenti normali, anche se questi li possono usare se necessario e se hanno i permessi.

`/etc`
: File di configurazione specifici della macchina.

`/root`
: La home directory dell'utente root.

`/lib`
: Le librerie condivise necessarie ai programmi sul filesystem radice.

`/lib/modules`
: I moduli caricabili del kernel, specialmente quelli che sono necessari per avviare il sistema quando lo si sta recuperando da un disastro (ad esempio i driver di rete e dei filesystem).

`/dev`
: I file di device.

`/tmp`
: I file temporanei. I programmi che vengono avviati dopo il boot dovrebbero usare `/var/tmp`, non `/tmp`, dato che il primo si trova probabilmente in un disco con più spazio.

`/boot`
: I file usati dal bootstrap loader, cioè LILO. Le immagini del kernel spesso vengono tenute qui invece che nella directory principale. Se ce ne è più di una, la directory può facilmente crescere parecchio, e spesso può essere meglio tenerla in un filesystem separato, anche per assicurarsi che le immagini del kernel siano comprese nei primi 1024 cilindri di un disco IDE.

`/mnt`
: Il punto di mount dove l'amministratore di sistema possa montare temporaneamente delle directory. I programmi non dovrebbero montarsi su `/mnt` automaticamente. `/mnt` può essere diviso in sottodirectory (ad

esempio `/mnt/dosa` può essere il floppy che usa un filesystem MS-DOS, e `/mnt/exta` lo stesso con un filesystem ext2).

`/proc, /usr, /var, /home`
Punti di mount per gli altri filesystem.

3.3 La directory `/etc`

La directory `/etc` contiene moltissimi file: alcuni di essi sono descritti qui sotto, per gli altri dovreste determinare a quale programma appartengono e leggere la pagina man corrispondente. Anche molti file di configurazione per la rete si trovano in `/etc`, e sono descritti nella *Guida dell'amministratore di rete*.

`/etc/rc` **o** `/etc/rc.d` **o** `/etc/rc?.d`
Script o directory di script da eseguire all'avvio o quando si cambia il runlevel. Vedere la Sezione su **init** per altre informazioni.

`/etc/passwd`
Il database degli utenti, con dei campi che danno il nome dell'utente, il vero nome, la home directory, la password criptata ed altre informazioni su ciascun utente. Il formato è documentato nella pagina man di **passwd**.

`/etc/fdprm`
La tabella dei parametri dei floppy disk. Descrive le caratteristiche di diversi formati di floppy e viene usata da **setfdprm**; vedere la pagina man di quest'ultimo per altre informazioni.

`/etc/fstab`
Elenca i filesystem montati automaticamente all'avvio dal comando **mount -a** (in `/etc/rc` o in un file di avvio equivalente). Sotto Linux contiene anche delle informazioni sulle aree di swap usate automaticamente da **swapon -a**. Vedere la Sottosezione *Montare e smontare un filesystem* nel Capitolo 4 e la pagina man di **mount** per ulteriori informazioni.

`/etc/group`
: Simile a `/etc/passwd`, ma descrive i gruppi anziché gli utenti. Vedere la pagina man di **group** per altre informazioni.

`/etc/inittab`
: File di configurazione di **init**.

`/etc/issue`
: L'output di **getty** prima del prompt di login. Di solito contiene una breve descrizione del sistema o un messaggio di benvenuto, la scelta del quale è lasciata all'amministratore di sistema.

`/etc/magic`
: Il file di configurazione di **file**; contiene la descrizione di vari formati di file, basandosi sui quali **file** indovina il tipo di file. Vedere la pagina man di `magic` e **file** per altre informazioni.

`/etc/motd`
: Il messaggio del giorno, visualizzato subito dopo ogni login riuscito. Il contenuto è a scelta dell'amministratore di sistema, e spesso viene usato per dare delle informazioni agli utenti, come ad esempio l'avviso degli orari di spengimento del sistema.

`/etc/mtab`
: Elenco dei filesystem montati al momento; viene impostato inizialmente dagli script di avvio, aggiornato automaticamente dal comando **mount** ed usato quando c'è bisogno di un elenco dei filesystem montati, come per il comando **df**.

`/etc/shadow`
: Il file delle shadow password sui sistemi in cui sono installate. Le shadow password spostano le password criptate da `/etc/passwd` a `/etc/shadow`; quest'ultimo non è leggibile da nessuno tranne root, in modo che sia più difficile craccare le password.

`/etc/login.defs`
: File di configurazione del comando **login**.

`/etc/printcap`
: Come `/etc/termcap`, ma per le stampanti; ha una sintassi diversa.

`/etc/profile`, `/etc/csh.login`, `/etc/csh.cshrc`
: File eseguiti al login o all'avvio dalle shell di tipo Bourne o C, permettono all'amministratore di sistema di impostare dei valori di default globali per tutti gli utenti. Vedere le pagine man per le rispettive shell.

`/etc/securetty`
: Identifica i terminali sicuri, cioè quelli da cui root può accedere al sistema. Tipicamente vengono elencate solo le console virtuali, in modo che sia impossibile (o almeno più difficile) acquistare i privilegi di superutente craccando un sistema da un modem o dalla rete.

`/etc/shells`
: Elenca le shell considerate sicure. Il comando **chsh** permette agli utenti di cambiare la loro shell di login solo con le shell elencate in questo file. **ftpd**, il processo server che dà i servizi FTP per la macchina controllerà che la shell dell'utente sia elencata in `/etc/shells` e non permetterà di collegarsi a chi ne usa una non in elenco.

`/etc/termcap`
: Il database dei terminali, descrive con quali "sequenze di escape" si controllano i vari terminali. I programmi vengono scritti in modo che invece di mandare in output una sequenza di escape che funzioni solo su una particolare marca di terminali, cercano la sequenza corretta per fare quello che vogliono fare in `/etc/termcap`. Come risultato la maggior parte dei programmi funzionano con quasi tutti i tipi di terminali. Vedere le pagine man di `termcap`, `curs_termcap` e `terminfo` per altre informazioni.

3.4 La directory `/dev`

La directory `/dev` contiene degli speciali file di device per i vari dispositivi. I file di device vengono chiamati usando delle convenzioni speciali, che sono descritte nell'elenco dei dispositivi (*Device list*, vedi [8]). I file di device vengono creati durante l'installazione, ma lo si può fare anche in seguito usando lo script **/dev/MAKEDEV**. **/dev/MAKEDEV.local** è uno script scritto dall'amministratore di sistema che crea file di device o collegamenti solo locali (cioè quelli che non fanno parte dello standard **MAKEDEV**, come i file di device per i driver di alcuni dispositivi non standard).

3.5 Il filesystem `/usr`

Il filesystem `/usr` è spesso grande, dato che vi sono installati tutti i programmi. Tutti i file in `/usr` vengono di solito da una distribuzione di Linux; i programmi installati in locale ed il resto vanno sotto `/usr/local`: in questo modo è possibile aggiornare il sistema ad una nuova versione della distribuzione, o addirittura ad una distribuzione completamente nuova, senza dover reinstallare tutti i programmi da capo. Alcune delle sottodirectory di `/usr` sono elencate qui sotto (alcune di quelle meno importanti sono state saltate, vedere il FSSTND [15]per altre informazioni).

`/usr/X11R6`
: Il sistema X Window, tutti i suoi file. Per semplificare lo sviluppo e l'installazione di X i suoi file non sono stati integrati nel resto del sistema, ma c'è un albero di directory sotto `/usr/X11R6` simile a quello sotto `/usr` stesso.

`/usr/X386`
: Simile a `/usr/X11R6`, ma per X11 Release 5.

`/usr/bin`
: Quasi tutti i comandi utente. Alcuni sono in `/bin` o in `/usr/local/bin`.

`/usr/sbin`
: I comandi di amministrazione di sistema che non sono necessari sotto il filesystem radice, come la maggior parte dei programmi di server.

`/usr/man`, `/usr/info`, `/usr/doc`
: Rispettivamente le pagine man, la documentazione info della GNU ed altri file di documentazione.

`/usr/include`
File di header per il linguaggio di programmazione C. Per essere consistenti con il resto del filesystem si sarebbero dovuti trovare in `/usr/lib` ma la tradizione è del tutto in favore di questo nome.

`/usr/lib`
File di dati che non vengono modificati per i programmi ed i sottosistemi, tra cui anche alcuni file di configurazione validi per tutto il sistema. Il nome `lib` viene da libreria: originariamente in `/usr/lib` erano memorizzate le librerie delle subroutine di programmazione.

`/usr/local`
Il posto per il software installato in locale e per gli altri file.

3.6 Il filesystem `/var`

`/var` contiene i dati che vengono modificati quando il sistema lavora normalmente. È specifico per ciascun sistema, cioè non viene condiviso in rete con altri computer.

`/var/catman`
Una cache per le pagine man che vengono formattate a richiesta. I sorgenti delle pagine man vengono di solito immagazzinati in `/usr/man/man*`; alcune hanno una versione preformattata che si trova in `/usr/man/cat*`, altre devono essere formattate la prima volta che vengono visualizzate; la versione formattata viene posta in `/var/man` in modo che chi le volesse visualizzare di nuovo non debba ripetere l'operazione (`/var/catman` viene ripulita spesso nello stesso modo delle directory temporanee).

`/var/lib`
I file che vengono modificati mentre il sistema viene normalmente usato.

`/var/local`
I dati variabili per i programmi che vengono installati in `/usr/local` (cioè quelli che sono stati installati dall'amministratore di sistema). Notate che anche i programmi installati in locale dovrebbero usare le altre directory `/var` se sono appropriate, ad esempio `/var/lock`.

`/var/lock`
: File di lock. Molti programmi seguono una convenzione per creare dei file di lock in `/var/lock` per indicare che stanno usando un particolare dispositivo o file; altri programmi noteranno il lock e non accetteranno di usare il dispositivo o il file in questione.

`/var/log`
: I file di log di vari programmi, specialmente di **login** (`/var/log/wtmp`, dove vengono memorizzati tutti i login e i logout del sistema) e di **syslog** (`/var/log/messages`, dove vengono immagazzinati tutti i messaggi del kernel e dei programmi di sistema). I file in `/var/log` possono spesso crescere indefinitamente, e possono richiedere che la directory venga ripulita ad intervalli regolari.

`/var/run`
: File che contengono informazioni sul sistema valide fino al successivo reboot. Ad esempio, `/var/run/utmp` contiene informazioni sugli utenti collegati in quel momento.

`/var/spool`
: Directory per le mail, le news, le code di stampa ed altre code. Ciascuno spool diverso ha la sua sottodirectory sotto `/var/spool`, ad esempio le caselle di posta degli utenti sono in `/var/spool/mail`.

`/var/tmp`
: File temporanei troppo grandi o che devono esistere per un tempo più lungo di quello permesso per `/tmp` (anche se l'amministratore di sistema può non permettere file molto vecchi neanche in `/var/tmp`).

3.7 Il filesystem `/proc`

Il filesystem `/proc` contiene un filesystem virtuale: non esiste sul disco, ma viene creato nella memoria dal kernel. Viene usato per fornire informazioni sul sistema (originariamente lo faceva sui processi, da cui il suo nome). Alcuni dei file e delle directory più importanti in esso contenute sono descritte qui sotto; maggiori dettagli si possono trovare nella pagina man di `proc`.

`/proc/1`
: Una directory con le informazioni sul processo numero 1. Ciascun processo ha una directory sotto `/proc` indicata dal numero di identificazione del processo.

`/proc/cpuinfo`
: Informazioni sul processore, come il tipo, la marca, il modello e la performance.

`/proc/devices`
: Un elenco dei device driver configurati nel kernel in uso.

`/proc/dma`
: I canali DMA usati al momento.

`/proc/filesystems`
: I filesystem configurati nel kernel.

`/proc/interrupts`
: Gli interrupt in uso e quanti ne sono stati chiamati di ciascuno.

`/proc/ioports`
: Le porte di I/O in uso al momento.

`/proc/kcore`
: Un'immagine della memoria fisica del sistema, della stessa dimensione di quest'ultima; non occupa così tanto spazio, ma viene generata in tempo reale quando i programmi vi accedono (ricordate: a meno che non lo copiate da qualche altra parte, nessun file sotto `/proc` occupa spazio disco).

`/proc/kmsg`
: Messaggi di output del kernel. Vengono mandati anche a **syslog**.

`/proc/ksyms`
: La tabella dei simboli per il kernel.

`/proc/loadavg`
: La media del carico (*load average*) del sistema: tre indicatori senza significato di quanto lavoro fa il sistema in ogni momento.

`/proc/meminfo`
: Informazioni sull'uso della memoria, sia fisica che di swap.

`/proc/modules`
: Quali moduli del kernel sono caricati al momento.

`/proc/net`
: Informazioni di stato sui protocolli di rete.

`/proc/self`
: Un link simbolico alla directory dei processi del programma che sta guardando in `/proc`. Quando due processi guardano in `/proc`, ottengono link diversi. Serve più che altro per facilitare ai programmi il raggiungimento della loro directory dei processi.

`/proc/stat`
: Varie statistiche sul sistema, come il numero di page fault da quando è stato avviato.

`/proc/uptime`
: Il tempo per cui il sistema è stato attivo.

`/proc/version`
: La versione del kernel.

Notate che mentre i file descritti qui sopra tendono ad essere file di testo leggibili facilmente, altri possono essere formattati in modo da non essere facilmente digeribili. Ci sono molti comandi che fanno poco più che leggere questi ultimi e formattarli in modo da poterli capire meglio. Ad esempio, il programma **free** legge `/proc/meminfo` e converte le quantità date in byte a kilobyte (aggiungendo anche altre informazioni).

Capitolo 4

Uso dei dischi e di altre memorie di massa

On a clear disk you can seek forever.

(Battuta intraducibile)

Quando si installa o si aggiorna un sistema, si lavora molto sui dischi: bisogna crearvi dei filesystem in modo da poterci memorizzare dei file e riservare spazio per le varie parti del sistema.

Questo capitolo spiega tutte queste attività iniziali. Di solito, una volta impostato il sistema, non dovrete ripetere queste operazioni, tranne che per i floppy: dovrete ritornare a questo capitolo solo per aggiungere un nuovo disco o per rifinire l'utilizzo di quelli che avete.

I compiti di base nell'amministrazione dei dischi sono:

- La formattazione, che fa diverse cose per preparare il disco all'uso, come controllare se ci sono settori danneggiati (oggigiorno non è necessario formattare la maggior parte degli hard disk).

- Il partizionamento, se volete usare il disco per attività diverse che non devono interferire l'una con l'altra. Dovete ripartizionare se volete usare diversi sistemi operativi sullo stesso disco, o se volete tenere i file degli utenti separati da quelli di sistema, cosa che aiuta a proteggere questi ultimi e semplifica i backup.

- La creazione di un filesystem (di tipo adatto) su ciascun disco o partizione. I dischi non risultano a Linux finché non ci create un filesystem: a quel punto potete memorizzarci file ed accedere a quelli presenti.

- Il montaggio (mount) di diversi filesystem per formare una singola struttura ad albero, sia automaticamente che manualmente, a seconda delle circostanze (dovete smontare a mano i filesystem montati manualmente).

Il Capitolo 5 contiene informazioni sulla memoria virtuale e sulla cache del disco, di cui dovete essere coscienti quando usate i dischi.

4.1 Due tipi di dispositivi

UNIX, e quindi Linux, riconosce due tipi diversi di dispositivi: quelli a blocchi ad accesso casuale, come i dischi, e quelli a carattere, come i nastri e le linee seriali, alcuni dei quali sono seriali, altri ad accesso casuale. Ogni dispositivo supportato viene rappresentato nel filesystem come un *file di device*. Quando si legge o si scrive un file di device, i dati provengono o vanno al dispositivo che esso rappresenta; in questo modo per accedere ai dispositivi non è necessario nessun programma speciale (e nessuna metodologia speciale per la programmazione delle applicazioni, come la cattura degli interrupt o il mandare richieste ad una porta seriale); ad esempio, per mandare un file alla stampante, basta dire

```
$ cat nomefile > /dev/lp1
$
```

ed il contenuto del file viene stampato (il file deve, naturalmente, essere in una forma riconoscibile dalla stampante). Comunque, dato che non è una buona idea avere diverse persone che inviano file usando **cat** contemporaneamente alla stampante, di solito si usa un programma speciale per farlo (normalmente **lpr**); un programma del genere si assicura che venga stampato solo un file alla volta e manderà gli altri in stampa in sequenza a mano a mano che la stampante ne finisce uno. Anche per la maggior parte degli altri dispositivi serve una cosa del genere; in effetti normalmente non ci si deve preoccupare per niente di questo tipo di file.

Dato che i dispositivi appaiono come file nel filesystem (nella directory `/dev`), è facile vedere quali file di device esistono usando **ls** o un altro comando adatto. Nell'output di **ls -l** la prima colonna contiene il tipo di file ed i suoi permessi. Ad esempio, per un dispositivo seriale sul mio sistema appare:

```
$ ls -l /dev/cua0
crw-rw-rw-   1 root     uucp       5,  64 Nov 30  1993 /dev/cua0
$
```

Il primo carattere della prima colonna, ad esempio '`c`' in `crw-rw-rw-` qui sopra, indica agli utenti informati il tipo del file, in questo caso un dispositivo a caratteri. Per i file ordinari il primo carattere è '`-`', per le directory è '`d`', e per i dispositivi a blocchi '`b`'; leggete la pagina man di **ls** per altre informazioni.

Notate che di solito esistono tutti i file di device anche se il dispositivo corrispondente può non essere installato, quindi se avete un file `/dev/sda` non è detto che abbiate realmente un hard disk SCSI. Avere tutti i file di device rende più semplici i programmi di installazione e semplifica l'aggiunta di nuovo hardware (non c'è bisogno di trovare i parametri corretti e creare il file di device per il nuovo dispositivo).

4.2 Gli hard disk

In questa Sezione viene introdotta la terminologia relativa agli hard disk: se già conoscete i termini ed i concetti potete saltarla.

In Figura 4.1 viene riportato un disegno schematico delle parti più importanti di un hard disk; questo consiste di una o più *piastre* circolari[1] con entrambe le *superfici* ricoperte da una sostanza magnetica usata per la memorizzazione dei dati. Per ciascuna superficie c'è una *testina di lettura e scrittura* che esamina o altera i dati memorizzati. Le piastre ruotano su un asse comune, con una velocità tipica di 3600 rotazioni al minuto, anche se gli hard disk ad alte prestazioni raggiungono velocità maggiori. Le testine si spostano lungo

[1] Le piastre sono fatte di un materiale duro, come l'alluminio, da cui il nome hard disk, cioè "disco duro".

il raggio della piastra e questo movimento, combinato con la rotazione della piastra stessa, permette alla testina di accedere a tutti i punti delle superfici.

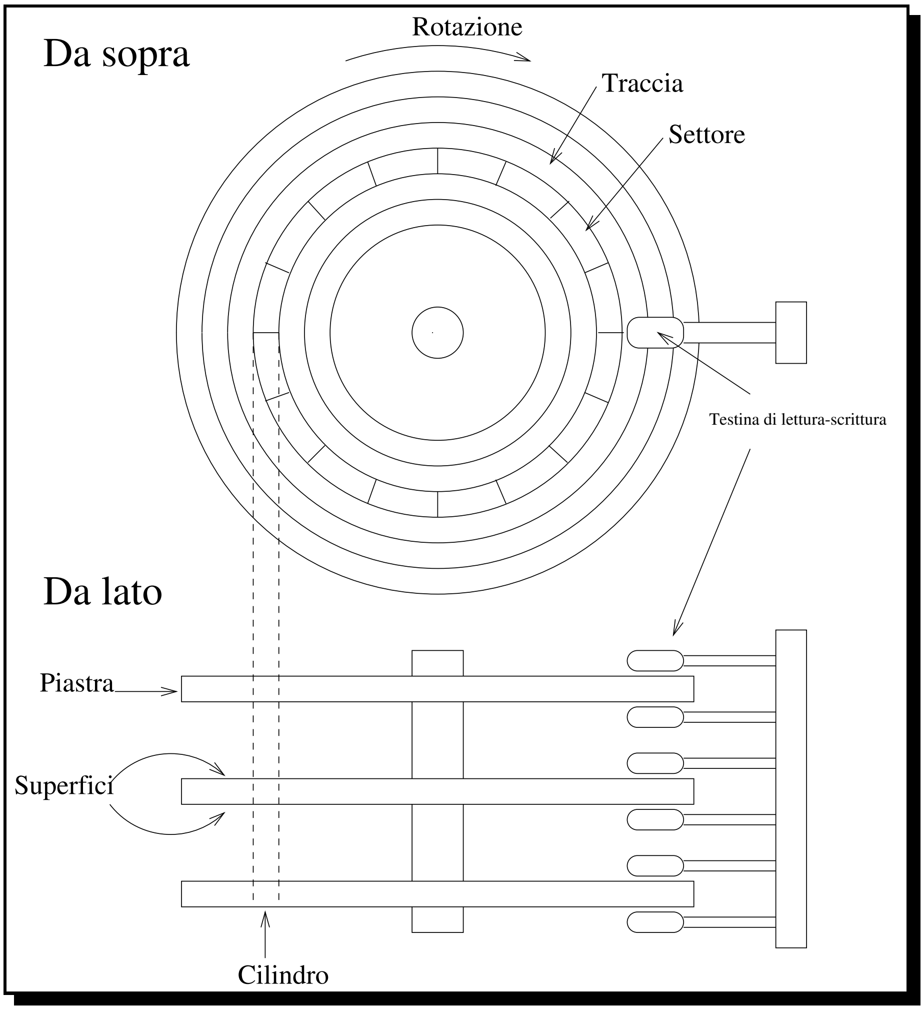
Il processore (CPU) e il disco in sé comunicano attraverso un *controller* che solleva il resto del computer dalla necessità di sapere come utilizzare l'hard disk, dato che si possono fare diversi tipi di controller per diversi tipi di disco che usino la stessa interfaccia verso il resto del computer. Quindi il computer può dire semplicemente "hey disco, dammi quello che voglio", invece di una serie lunga e complessa di segnali elettrici per spostare la testina al punto desiderato, aspettare che la parte giusta ruoti sotto la testina e tutte queste cose antipatiche (in realtà, l'interfaccia al controller è comunque complessa, ma molto meno di come sarebbe dovuta essere altrimenti). Il controller può anche fare altre cose, come gestire la cache o isolare automaticamente i settori danneggiati.

Di solito è sufficiente sapere questo sull'hardware; in realtà ci sono anche molte altre cose, come il motore che fa girare le piastre e sposta le testine e l'elettronica che controlla le operazioni delle parti meccaniche, ma tutto questo non è rilevante per la comprensione del principio di funzionamento di un hard disk.

Le superfici sono di solito divise in anelli concentrici, chiamati *tracce*, e queste a loro volta sono divise in *settori*. Questa divisione viene usata per specificare punti dell'hard disk e per allocare lo spazio disco per i file. Per trovare un punto determinato dell'hard disk si potrebbe dire "piastra 3, traccia 5, settore 7". Di solito il numero dei settori è lo stesso per tutte le tracce, ma in alcuni hard disk ci sono più tracce nei settori più esterni (tutti i settori sono delle stesse dimensioni fisiche, quindi nelle tracce più esterne ce ne entrano di più). Tipicamente, un settore conterrà 512 byte di dati. Il disco stesso non può gestire quantità di dati più piccole di un settore.

Ciascuna superficie è divisa in tracce (e settori) nello stesso modo. Ciò significa che quando la testina di una superficie è su una traccia, quella dell'altra superficie è anch'essa sulla traccia corrispondente. Tutte le tracce corrispondenti prese insieme si chiamano *cilindro*. Ci vuole del tempo per spostare le testine da una traccia (cilindro) ad un'altra, quindi, mettendo i dati a cui si accede in contemporanea (diciamo un file) in modo che si trovino tutti all'interno dello stesso cilindro, non è necessario spostare la testina per leggerli tutti e viene migliorata la performance del disco. Non è sempre possibile mettere i

Figura 4.1 – *Schema di un hard disk.*



file in questo modo: i file che vengono immagazzinati in posti diversi dell'hard disk vengono chiamati *frammentati*.

Il numero di superfici (o testine, che è la stessa cosa), cilindri e settori variano molto: le specifiche sul numero di ciascuno di questi elementi vengono chiamate *geometria* di un hard disk. La geometria viene di solito memorizzata in una locazione di memoria speciale, alimentata a batterie, chiamata *RAM CMOS*, da cui il sistema operativo la può leggere durante l'avvio del computer o l'inizializzazione dei driver.

Sfortunatamente, il BIOS[2] ha una limitazione che rende impossibile specificare un numero di tracce più alto di 1024 nella RAM CMOS, che è troppo poco per un hard disk grande. Per superare questo problema il controller degli hard disk mente sulla geometria e *traduce gli indirizzi* dati dal computer in qualcosa di adatto. Ad esempio, un hard disk può avere 8 testine, 2048 tracce e 35 settori per traccia[3]. Il suo controller può mentire al computer e dire che ha 16 testine, 1024 tracce e 35 settori per tracce, non superando così il limite per le tracce, e tradurre gli indirizzi che il computer gli dà dimezzando il numero della testina e raddoppiando quello della traccia. La matematica può in realtà essere molto più complicata perché i numeri in genere non sono così carini, ma i dettagli non sono rilevanti per la comprensione del principio. Questa traduzione distorce la visione del sistema operativo di come è organizzato il disco, rendendo poco pratico usare il trucco di mettere tutti i dati su uno stesso cilindro per aumentare la performance.

Questa traduzione è un problema solo per i dischi IDE; quelli SCSI usano un numero di settori sequenziale (cioè il controller traduce un numero di settore sequenziale nella tripletta testina, cilindro e settore) ed un metodo totalmente diverso per la comunicazione tra la CPU e il controller, quindi il problema non si pone. Notate, comunque, che il computer può non sapere la geometria reale neanche di un disco SCSI.

Dato che Linux spesso non conosce la geometria reale di un disco, i suoi filesystem non provano neanche a mantenere i file all'interno di un singolo cilindro; provano invece ad assegnare ai file settori numerati sequenzialmente, cosa che dà quasi sempre una performance simile. La questione è complicata

[2] Il BIOS è un software fisso immagazzinato nei chip della ROM. Ha il compito, tra le altre cose, di gestire gli stadi iniziali del processo di avvio.

[3] I numeri sono completamente inventati.

ulteriormente dalle cache sul controller e dai precaricamenti automatici fatti dal controller stesso.

Ciascun hard disk è rappresentato da un file di device separato. Di solito ci possono essere solo due o quattro hard disk IDE, che corrispondono rispettivamente a `/dev/hda`, `/dev/hdb`, `/dev/hdc`, e `/dev/hdd`. Gli hard disk SCSI corrispondono a `/dev/sda`, `/dev/sdb`, e così via. Esistono delle convenzioni simili anche per altri tipi di hard disk: per altre informazioni vedere [8]. Notare che i file di device per gli hard disk danno accesso al disco intero, senza considerare le partizioni (che verranno spiegate più avanti) ed è facile rovinare i dati in essi contenuti se non si fa attenzione. I file di device dei dischi di solito vengono usati solo per accedere al loro master boot record (che verrà anch'esso spiegato più avanti).

4.3 I floppy

Un floppy disk consiste di una membrana flessibile coperta su uno solo o su entrambi i lati con una sostanza magnetica simile a quella degli hard disk. Il floppy in sé non ha una testina di lettura e scrittura, ma questa è inclusa nel lettore. Un floppy corrisponde ad una piastra di un hard disk, ma è rimuovibile ed un lettore può essere usato per accedere a diversi floppy, mentre l'hard disk è una singola unità indivisibile.

Come un hard disk, un floppy è suddiviso in tracce e settori (e le due tracce corrispondenti sui due lati di un floppy formano un cilindro), ma ne ha meno di un hard disk.

Un lettore di floppy può di solito usare diversi tipi di dischi: ad esempio, un lettore da 3.5 pollici può usare sia dischi da 720 kB che da 1.44 MB; dato che il lettore deve operare in maniera leggermente diversa e che il sistema operativo deve conoscere la dimensione del disco, ci sono più file di device per ciascun lettore di floppy, uno per ciascuna combinazione di lettore e tipo di disco. Quindi, `/dev/fd0H1440` è il primo lettore di floppy (fd0), un lettore a 3.5 pollici, che usa un dischetto da 3.5 pollici ad alta densità (H) di dimensioni 1440 kB (1440), cioè un normale floppy da 3.5 HD. Per altre informazioni sulle convenzioni per i nomi dei dispositivi floppy, vedere [8].

I nomi dei device dei floppy sono complessi, comunque, quindi Linux ne ha uno che individua automaticamente il tipo di disco nel lettore; funziona provando a leggere i primi settori di un dischetto inserito nel lettore usando diversi tipi di floppy finché trova quello corretto. Naturalmente ciò richiede che il dischetto sia prima stato formattato. I device automatici sono `/dev/fd0`, `/dev/fd1`, e così via.

I parametri che il device automatico usa per accedere ad un disco possono essere impostati anche usando il programma **setfdprm**. Una cosa del genere può essere utile se dovete usare dei dischi che non seguono le normali dimensioni dei floppy: ad esempio se hanno un numero di settori non usuale, o se per qualche ragione il riconoscimento automatico non funziona e manca il corrispondente file di device.

Linux può gestire molti formati non standard di floppy in aggiunta a tutti quelli standard, anche se alcuni di questi richiedono di usare dei programmi di formattazione speciali. Non vedremo questi tipi di disco ora, ma nel frattempo potete esaminare il file `/etc/fdprm`, che specifica le impostazioni riconosciute da **setfdprm**.

Il sistema operativo deve sapere quando viene cambiato il disco nel drive dei floppy, ad esempio per evitare di usare dati messi in cache dal disco precedente. Sfortunatamente, la linea di segnale che viene usata per questo qualche volta non funziona e, peggio, non sarà sempre individuabile quando il lettore viene usato da MS-DOS. Se avete degli strani problemi quando usate i floppy, forse la ragione è questa. L'unico modo di correggere questo problema è riparare il lettore dei floppy.

4.4 I CD-ROM

I lettori di CD-ROM usano dischi a lettura ottica ricoperti di plastica. Le informazioni vengono registrate sulla superficie del disco[4] in piccoli 'fori' allineati lungo una spirale che va dal centro verso l'esterno. Il lettore dirige un raggio laser lungo la spirale per leggere il disco. Quando il laser colpisce un buco, il raggio viene riflesso in un modo, quando colpisce la superficie liscia viene riflesso in un altro, rendendo facile codificare i bit e quindi le informazioni. Il resto è facile, semplice meccanica.

I lettori di CD-ROM sono lenti se paragonati agli hard disk. Mentre un tipico hard disk ha un tempo di ricerca medio minore di 15 millisecondi, un

[4] Cioè la superficie di metallo del disco, all'interno della copertura di plastica.

CD-ROM veloce si attesta sui decimi di secondo; una velocità di trasferimento dati reale di centinaia di kilobyte al secondo è piuttosto alta. La lentezza sta a significare che i lettori di CD-ROM non sono comodi da usare al posto degli hard disk (alcune distribuzioni di Linux forniscono filesystem 'live' su CD-ROM, rendendo inutile copiare i file sull'hard disk e quindi facilitando l'installazione e salvando molto spazio su disco), anche se è sempre possibile. Per installare nuovo software i CD-ROM sono molto validi, perché non è essenziale una velocità altissima durante l'installazione.

Ci sono diversi modi di sistemare i dati su un CD-ROM; il modo più comune è specificato dallo standard internazionale ISO 9660. Questo standard specifica un filesystem minimale, anche più povero di quello usato dall'MS-DOS; d'altra parte, è talmente minimale che qualsiasi sistema operativo dovrebbe essere in grado di mapparlo sul suo sistema nativo.

Per l'uso normale sotto UNIX non si può usare il filesystem ISO 9660, quindi è stata sviluppata un estensione allo standard: le estensioni Rock Ridge. Le Rock Ridge permettono di usare nomi più lunghi, link simbolici e molte altre caratteristiche, rendendo un CD-ROM molto simile ad un qualsiasi filesystem UNIX moderno. Ancora meglio, un filesystem Rock Ridge è ancora un filesystem ISO 9660 valido e quindi può essere usato anche su sistemi non UNIX. Linux supporta sia l'ISO 9660 che le estensioni Rock Ridge e queste vengono riconosciute ed utilizzate automaticamente.

Il filesystem è però solo metà del lavoro: la maggior parte dei CD-ROM contengono dati che richiedono programmi speciali per potervi accedere e la maggior parte di questi programmi non girano sotto Linux (eccetto forse sotto `dosemu`, l'emulatore di MS-DOS per Linux).

Si accede ai lettori di CD-ROM attraverso il corrispondente file di device. Ci sono diversi modi di connetterne uno al computer: attraverso un controller SCSI, una scheda audio o un controller EIDE. Il montaggio dell'hardware va al di fuori dello scopo di questo libro, ma il tipo di connessione implica quale file di device debba essere usato. Vedere [8] per avere chiarimenti.

4.5 Le unità nastro

I lettori nastro usano dei nastri simili[5] alle cassette usate per la musica. Un nastro è di sua natura seriale, cioè per accedere ad una sua qualsiasi parte bisogna scorrere tutto quello che c'è in mezzo; un disco è un dispositivo ad accesso casuale, cioè si può passare direttamente ad un suo punto qualsiasi. L'accesso seriale dei nastri li rende dispositivi lenti; d'altra parte i nastri sono relativamente a buon mercato, dato che non hanno bisogno di essere veloci, possono avere anche una lunghezza elevata, e quindi contenere grandi quantità di dati: tutto questo li rende molto adatti a cose come l'archiviazione e i backup, che non richiedono velocità elevate ma traggono vantaggio dal basso costo e dalla grande capacità di memorizzazione.

4.6 La formattazione

La *formattazione* è il processo con cui si segnano sul mezzo magnetico le tracce ed i settori. Prima che venga formattato il disco ha una superficie in cui i segnali magnetici si sovrappongono in maniera caotica, mentre dopo la formattazione si porta un certo ordine nel caos, essenzialmente tracciando delle linee dove vanno le tracce e dove esse vengono suddivise in settori. I dettagli reali non sono proprio questi, ma è irrilevante; quello che è importante è che un disco non può essere usato se non è stato formattato.

La terminologia a questo proposito è piuttosto confusa: nell'MS-DOS, la parola formattazione viene usata anche per il processo di creazione di un filesystem (che verrà discusso più avanti); in questo caso i due processi sono spesso combinati, specialmente per i floppy. Quando sarà necessaria una distinzione, la formattazione reale viene chiamata *formattazione a basso livello*, mentre la creazione del filesystem si chiama *formattazione ad alto livello*. Nei circoli UNIX, i due processi vengono chiamati *formattazione* e *creazione di un filesystem*, quindi in questo libro verranno usati questi termini.

Per i dischi IDE ed alcuni dischi SCSI la formattazione viene fatta in realtà in fabbrica e non c'è bisogno di ripeterla, quindi la maggior parte delle persone non avranno bisogno di preoccuparsene. In effetti, formattare un hard disk può provocarne un malfunzionamento, perché può essere necessario farlo

[5] Ma del tutto diversi, naturalmente.

in maniera particolare per permettere la sostituzione automatica dei settori danneggiati.

I dischi che devono o possono essere formattati spesso richiedono comunque un programma speciale, dato che l'interfaccia alla logica di formattazione all'interno del disco è diversa da caso a caso. Il programma di formattazione spesso è sul BIOS del controller o viene fornito come programma MS-DOS; in nessuno dei due casi può essere usato facilmente da Linux.

Durante la formattazione si possono incontrare dei punti danneggiati sul disco, che vengono chiamati *blocchi danneggiati* o *settori danneggiati*; qualche volta vengono gestiti dallo stesso drive, ma anche in questi casi, se se ne sviluppano altri, bisogna fare qualcosa per evitare di usare quelle parti del disco. La logica per farlo è interna al filesystem; come aggiungervi le informazioni utili viene descritto qui sotto. In alternativa si può creare una piccola partizione che ricopra solo la parte danneggiata del disco: può essere una buona idea se questa è piuttosto grande, dato che i filesystem possono avere dei problemi con aree danneggiate di grandi dimensioni.

I floppy vengono formattati con **fdformat**. Il file di device del floppy da usare viene dato come parametro; ad esempio, il seguente comando formatterebbe un dischetto ad alta densità, da 3. 5 pollici, nel primo lettore di floppy:

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track.  Total capacity 1440 kB.
Formatting ...  done
Verifying ...  done
$
```

Notate che se volete usare un dispositivo che venga rilevato automaticamente (ad esempio `/dev/fd0`), *dovete* per prima cosa impostarne i parametri con **setfdprm**. Per avere lo stesso effetto del comando qui sopra, bisognerebbe fare così:

```
$ setfdprm /dev/fd0 1440/1440
$ fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track.  Total capacity 1440 kB.
Formatting ...  done
Verifying ...  done
$
```

Di solito è più conveniente scegliere il file di device corretto che corrisponda al tipo di floppy; notate che non è saggio formattare i floppy ad una capacità maggiore di quella per cui sono stati progettati.

fdformat convaliderà anche il floppy, cioè controllerà che non ci siano blocchi danneggiati; proverà a leggere e scrivere su un blocco danneggiato diverse volte (lo si può sentire, dato che il rumore del lettore cambia moltissimo). Se il floppy è solo leggermente danneggiato (a causa dello sporco sulla testina del lettore alcuni errori sono segnali falsi), **fdformat** non si lamenterà, ma un errore reale farà abortire il processo di convalida. Il kernel stamperà dei messaggi di log per ogni errore di I/O che trova, messaggi che andranno in console o, se si usa **syslog**, nel file `/var/log/messages`. **fdformat** non dice dove si trova l'errore (di solito non ci interessa, i floppy sono abbastanza a buon mercato da poterne buttare uno automaticamente, se è danneggiato).

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ...  done
Verifying ...  read: Unknown error
$
```

Per cercare soltanto i blocchi danneggiati su un disco o su una partizione si usa il comando **badblocks** (anche per i floppy); non formatta il disco, quindi lo si può usare anche su filesystem esistenti. L'esempio qui sotto controlla un floppy da 3.5 pollici con due blocchi danneggiati.

```
$ badblocks /dev/fd0H1440 1440
718
719
$
```

badblocks rende in output i numeri dei blocchi che trova danneggiati. La maggior parte dei filesystem riesce ad evitarli, mantenendone un elenco che viene inizializzato quando viene creato il filesystem e può essere modificato in seguito. La prima ricerca dei blocchi danneggiati può essere fatta con il comando **mkfs** (che inizializza il filesystem), ma i controlli successivi vanno fatti con **badblocks** ed i nuovi blocchi dovrebbero essere aggiunti con **fsck**. Descriveremo **mkfs** e **fsck** più avanti.

Molti dischi moderni individuano automaticamente i blocchi danneggiati e tentano di recuperarli usando al loro posto un blocco speciale, integro, riservato per questo scopo, che non è visibile al sistema operativo. Questa caratteristica dovrebbe essere documentata nel manuale del disco, se siete curiosi di sapere se ce l'avete. Anche in dischi di questo tipo si può avere perdita di dati, se il numero di blocchi danneggiati diventa troppo grande, anche se è probabile che a quel punto il disco sarebbe troppo rovinato per essere utile.

4.7 Le partizioni

Un singolo hard disk può essere diviso in diverse *partizioni*, ciascuna delle quali funziona come se fosse un disco separato. L'idea è che se avete un hard disk e ad esempio volete avere due sistemi operativi, potete suddividere il disco in due partizioni; ciascun sistema operativo userà la sua partizione come vuole e non toccherà quella dell'altro. In questo modo i due sistemi possono coesistere in pace sullo stesso hard disk, mentre senza le partizioni ci sarebbe voluto un disco per ciascun sistema operativo.

I floppy non vengono partizionati, non per motivi tecnici, ma perché sono molto piccoli e creare al loro interno delle partizioni non sarebbe utile se non in casi molto rari. I CD-ROM di solito non vengono partizionati, dato che è molto più semplice usarli come un grande disco e in genere non è necessario avere diversi sistemi operativi sullo stesso CD-ROM.

4.7.1 L'MBR, i settori di boot e la tabella delle partizioni

Le informazioni sul partizionamento di un hard disk si trovano nel suo primo settore (cioè, il primo settore della prima traccia della prima superficie del disco). Questo settore si chiama *master boot record* (MBR) del disco: è il settore che il BIOS legge ed avvia quando la macchina viene accesa. Il master boot record contiene un piccolo programma che legge la tabella delle partizioni, controlla quale partizione è attiva (cioè quale è contrassegnata come avviabile) e legge il primo settore di quella partizione, il *boot sector* (settore di avvio) della partizione (anche l'MBR è un settore di avvio, ma ha uno status speciale e quindi un nome speciale). Il boot sector contiene un altro programmino che legge la prima parte del sistema operativo contenuto in quella partizione (sempre che sia avviabile) e lo avvia.

Lo schema di partizionamento non è costruito all'interno dell'hardware e nemmeno nel BIOS, è solo una convenzione che viene seguita da molti sistemi operativi: non tutti, ma quelli che non lo fanno sono le eccezioni. Alcuni sistemi operativi permettono l'uso di partizioni, ma occupano una partizione sull'hard disk ed usano il loro metodo di divisione specifico al suo interno. Quest'ultimo tipo coesiste pacificamente con gli altri sistemi operativi (incluso Linux) e non richiede misure speciali, ma un sistema operativo che non supporta le partizioni non può coesistere sullo stesso disco con un altro sistema.

Come precauzione è una buona idea scrivere su carta la tabella delle partizioni, in modo che se si corrompesse ci sarebbe una speranza di non perdere tutti i file (una tabella delle partizioni corrotta si può recuperare con **fdisk**). Le informazioni rilevanti si ricavano dal comando **fdisk -l**:

```
$ fdisk -l /dev/hda

Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes

   Device Boot  Begin   Start     End  Blocks    Id  System
/dev/hda1           1       1      24   10231+   82  Linux swap
/dev/hda2          25      25      48   10260    83  Linux native
/dev/hda3          49      49     408  153900    83  Linux native
/dev/hda4         409     409     790  163305     5  Extended
/dev/hda5         409     409     744  143611+   83  Linux native
/dev/hda6         745     745     790   19636+   83  Linux native
$
```

4.7.2 Partizioni estese e partizioni logiche

Lo schema di partizionamento originale degli hard disk dei PC permetteva solo quattro partizioni, ma presto questo si è dimostrato troppo poco per l'uso reale, perché alcune persone volevano più di quattro sistemi operativi (Linux, MS-DOS, OS/2, Minix, FreeBSD, NetBSD, o Windows NT, per nominarne alcuni), ma principalmente perché a volte è una buona idea avere diverse partizioni per un solo sistema operativo. Ad esempio per Linux è meglio mettere lo spazio di swap in una sua partizione, invece che in quella principale di Linux, per motivi di velocità (vedi sotto).

Per superare questo problema di progettazione, furono create le *partizioni estese*. Questo trucco permette di partizionare una *partizione primaria* in sotto-partizioni. La partizione primaria così suddivisa si dice estesa e le sotto-partizioni sono *partizioni logiche*: si comportano come primarie[6] ma vengono create in maniera diversa; non comportano una differenza di velocità.

La struttura delle partizioni di un hard disk può apparire come in Figura 4.2: il disco viene diviso in tre partizioni primarie, la seconda delle quali è divisa in due partizioni logiche, e parte del disco non viene partizionato. Il disco intero e ciascuna partizione primaria hanno un settore di boot.

4.7.3 Tipi di partizione

Le tabelle delle partizioni (quella nell'MBR, e quelle per le partizioni estese) contengono un byte per partizione che ne identifica il tipo. In questo modo si cerca di identificare il sistema operativo che usa la partizione o il modo in cui essa viene usata. Lo scopo è evitare che due sistemi operativi usino accidentalmente la stessa, comunque, in realtà, i sistemi operativi non considerano veramente il tipo della partizione: ad esempio, Linux proprio non lo legge. Peggio ancora, alcuni sistemi lo usano in maniera non corretta: ad esempio, almeno alcune versioni di DR-DOS ignorano il bit più significativo del byte, altre no.

Non c'è nessuna agenzia di standardizzazione che specifica cosa significhi ciascun valore di questo byte, ma alcuni valori comunemente accettati sono riportati in Tabella 4.1. La stessa lista è disponibile nel programma **fdisk** di Linux.

4.7.4 Ripartizionare un hard disk

Ci sono molti programmi per creare e rimuovere partizioni. La maggior parte dei sistemi operativi ne ha uno proprio ed è una buona idea usare quello del sistema che si usa, in caso faccia qualcosa di insolito che gli altri non fanno. La maggior parte di questi programmi, compreso quello per Linux, si chiamano **fdisk**, o variazioni sul tema. I dettagli sull'uso dell'**fdisk** di Linux sono riportati nella sua pagina man. Il comando **cfdisk** è simile a **fdisk**, ma ha un'interfaccia utente più carina (a tutto schermo).

Quando si usano dei dischi IDE la partizione di boot (quella con i file di immagine del kernel che viene avviato) deve essere completamente all'interno

[6] Illogiche?

Figura 4.2 – *Un esempio di partizionamento di hard disk.*

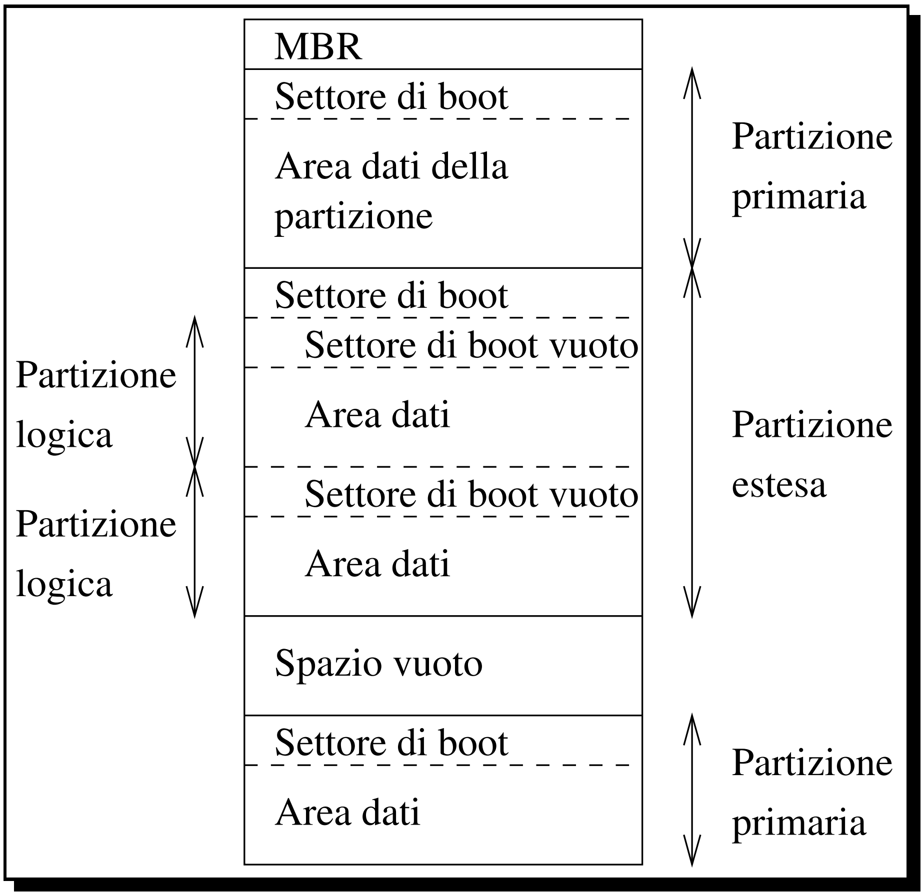


Tabella 4.1: Tipi di partizione (dal programma **fdisk** di Linux).

0	Vuota	40	Venix 80286	94	Amoeba BBT
1	DOS 12-bit FAT	51	Novell?	a5	BSD/386
2	XENIX root	52	Microport	b7	BSDI fs
3	XENIX usr	63	GNU HURD	b8	BSDI swap
4	DOS 16-bit <32M	64	Novell	c7	Syrinx
5	Estesa	75	PC/IX	db	CP/M
6	DOS 16-bit >=32M	80	MINIX vecchio	e1	DOS access
7	OS/2 HPFS	81	Linux/MINIX	e3	DOS sola lettura
8	AIX	82	Linux swap	f2	DOS secondaria
9	AIX avviabile	83	Linux nativa	ff	BBT
a	OS/2 Boot Manag	93	Amoeba		

dei primi 1024 cilindri, perché il disco viene usato dal BIOS durante l'avviamento del sistema (prima che il sistema vada in modalità protetta) e questo non ne può gestire di più. Talvolta è possibile usare una partizione di boot che è solo parzialmente entro i primi 1024 cilindri: questo trucco funziona solo se tutti i file che vengono letti dal BIOS sono entro il cilindro 1024. Dato che una cosa del genere è difficile da imporre, è *una pessima idea* provarci: non saprete mai quando un aggiornamento del kernel o la deframmentazione del disco vi toglierà la possibilità di avviare il sistema, quindi assicuratevi che la partizione di boot sia completamente all'interno dei primi 1024 cilindri.

Alcune versioni più recenti del BIOS e dei dischi IDE possono, in effetti, gestire dischi con un numero maggiore di cilindri. Se avete un sistema di questo genere, potete dimenticarvi del problema; se non ne siete sicuri, mettetela comunque entro i primi 1024.

Ciascuna partizione dovrebbe avere un numero pari di settori, dato che i filesystem di Linux usano blocchi di 1 kb, cioè due settori. Un numero dispari di settori comporterà il mancato utilizzo dell'ultimo; non crea problemi, ma è brutto ed alcune versioni di **fdisk** vi daranno un avvertimento.

Modificare la dimensione di una partizione di solito richiede di farne il backup (preferibilmente di tutto il disco, non si sa mai), cancellarla, crearne una nuova e rimetterci tutti i dati. Se la partizione deve aumentare, dovrete anche modificare la dimensione (e fare il backup e il restore) delle partizioni confinanti.

Dato che modificare la dimensione di una partizione è una cosa complicata, è preferibile farlo bene la prima volta o avere un mezzo di backup efficace e facile da usare. Se state installando da un mezzo che non richiede molto intervento umano (ad esempio un CD-ROM e non dei floppy), spesso è più facile giocare con diverse configurazioni all'inizio: dato che non avete dati di cui fare backup non è poi così doloroso modificare le dimensioni delle partizioni diverse volte.

Esiste un programma per l'MS-DOS, chiamato **fips**, che ridimensiona una partizione di MS-DOS o di Windows senza bisogno di dover fare backup e recuperare tutti i dati; per gli altri tipi di filesystem è però ancora necessario seguire tutta la procedura.

4.7.5 File di device e partizioni

Ciascuna partizione o partizione estesa ha il proprio file di device. Per convenzione i nomi di questi file sono composti dal nome del disco e dal numero della partizione, tenendo conto che da 1 a 4 si tratta di partizioni primarie (indipendentemente da quante partizioni primarie ci siano in realtà) e da 5 a 8 di partizioni logiche (indipendentemente da quale sia la partizione primaria che le contiene). Ad esempio, `/dev/hda1` è la prima partizione primaria sul primo disco IDE, e `/dev/sdb7` è la terza partizione estesa sul secondo disco SCSI. L'elenco dei dispositivi [8] dà altre informazioni su questo argomento.

4.8 I filesystem

4.8.1 Cosa sono i filesystem?

Un *filesystem* comprende i metodi e le strutture di dati usate da un sistema operativo per tenere traccia dei file su un hard disk o su una sua partizione, cioè il modo in cui i file sono organizzati sui dischi. La parola viene anche usata per riferirsi ad una partizione o ad un disco usato per immagazzinare i file, o al tipo del filesystem stesso, quindi si può dire "ho due filesystem" per indicare che si hanno due partizioni in cui immagazzinare file, o che si sta usando “il filesystem extended” per indicarne il tipo.

La differenza tra un disco o una partizione ed il filesystem che esso contiene è molto importante. Alcuni programmi (compreso, abbastanza ragionevolmente, quelli che creano i filesystem) operano direttamente sui settori del disco o della partizione; se c'è un filesystem esistente, questo verrà distrutto o seriamente danneggiato. La maggior parte dei programmi operano su un filesystem e quindi non funzioneranno su una partizione che non ne contiene uno (o che ne contiene uno del tipo sbagliato).

Prima che si possa usare un disco o una partizione come filesystem, questo deve essere inizializzato e bisogna scriverci le strutture di dati per l'archiviazione: questo processo si chiama *creazione di un filesystem.*

La maggior parte dei tipi di filesystem UNIX hanno una struttura generale simile, anche se i dettagli esatti cambiano abbastanza. I concetti centrali sono quelli di *superblocco*, *inode*, *blocco dati*, *blocco directory* e *blocco indirezione*. Il superblocco contiene delle informazioni sul filesystem intero, come la sua dimensione (l'informazione esatta qui dipende dal filesystem). Un inode contiene tutte le informazioni su un file eccetto il suo nome, che viene immagazzinato nella directory insieme con il numero dell'inode. Una voce di directory consiste del nome di un file e del numero dell'inode che lo rappresenta. L'inode contiene il numero di diversi blocchi dati, che vengono usati per immagazzinare i dati del file. Nell'inode c'è posto solo per alcuni numeri di blocchi dati, ma se ne servono di più viene allocato dinamicamente più spazio per i puntatori ai blocchi dati. Questi blocchi allocati dinamicamente si chiamano blocchi indiretti; il nome indica che per trovare i blocchi dati bisogna trovare prima il numero all'interno del blocco indiretto.

I filesystem UNIX di solito permettono di creare un *buco* in un file (si fa con **lseek**: controllate la pagina man), cioè il filesystem fa solo finta che in un posto particolare nel file ci sono solo byte zero, ma per quel posto nel file non viene riservato nessun settore di disco reale (cioè il file userà un po' meno spazio disco). Accade spesso solo per i file binari piccoli, le librerie condivise di Linux, alcuni database ed altri casi speciali (i buchi sono implementati immagazzinando un valore speciale come indirizzo del blocco dati nel blocco indiretto o nell'inode. Questo indirizzo speciale significa che non viene allocato nessun blocco dati per quella parte del file e quindi che questo ha un buco).

I buchi sono moderatamente utili. Sul sistema dell'autore una semplice misura ha mostrato un potenziale per circa 4 MB di risparmio usando buchi, su un totale di 200 MB di spazio disco usato. Quel sistema, però, contiene pochi programmi e nessun file di database.

4.8.2 Filesystem a go-go

Linux supporta diversi tipi di filesystem. Al momento in cui sto scrivendo, i più importanti sono:

minix
: Il più vecchio e si presume il più affidabile, ma piuttosto limitato (mancano alcuni time stamp, e i nomi di file sono al massimo di 30 caratteri) e di capacità ristrette (al massimo 64 MB per filesystem).

xia
: Una versione modificata del filesystem minix, che alza i limiti sulla lunghezza dei nomi di file e sulla dimensione dei filesystem, ma non introduce nuove caratteristiche. Non è molto conosciuto, ma si dice funzioni molto bene.

ext2
: Il più completo dei filesystem nativi di Linux ed al momento anche il più usato. È disegnato per essere compatibile in avanti, in modo che per usare nuove versioni del codice non si abbia bisogno di ricreare i filesystem esistenti.

ext
: Una versione più vecchia dell'ext2 che non era compatibile in avanti. Non è usato quasi mai in nuove installazioni, perché la maggior parte delle persone si sono convertite all'ext2.

Oltre a questi vengono supportati molti filesystem di altri sistemi operativi, per rendere più semplice lo scambio di file; questi filesystem funzionano come quelli nativi, tranne per la mancanza di alcune caratteristiche normali di quelli UNIX, oppure hanno delle limitazioni curiose o altre stranezze.

msdos
: Compatibile con i filesystem FAT di MS-DOS (ed OS/2 e Windows NT).

usmdos

Estende il driver msdos sotto Linux, in modo da avere i nomi di file lunghi, i permessi, i link e i file di device, e da assegnare ciascun file ad un utente. In questo modo è possibile usare un normale filesystem msdos come se fosse uno Linux, eliminando la necessità di avere una partizione separata per quest'ultimo.

iso9660

Il filesystem standard per i CD-ROM: viene supportata automaticamente l'estensione Rock Ridge allo standard per i CD-ROM, che permette di avere i nomi dei file lunghi.

nfs

Un filesystem di rete che permette la condivisione dei file tra vari computer per avervi un accesso più facile.

hpfs

Il filesystem di OS/2.

sysv

I filesystem SystemV/386, Coherent e Xenix.

La scelta del filesystem da usare dipende dalla situazione: se ci sono ragioni di compatibilità o altri motivi che rendono indispensabile l'uso di uno dei filesystem non nativi, deve essere usato quello, se si può scegliere liberamente allora probabilmente la scelta più saggia è l'ext2, dato che ha tutte le caratteristiche e la sua performance è la migliore.

Esiste anche il filesystem **proc**, di solito accessibile nella directory `/proc`, che in realtà non è per niente un filesystem anche se gli assomiglia. Il filesystem **proc** permette di avere facile accesso ad alcune strutture di dati del kernel, come la lista dei processi (da cui il nome); fa apparire queste strutture di dati come un filesystem, che può essere manipolato con i normali strumenti per i file. Ad esempio, per avere un elenco di tutti i processi si può usare il comando

```
$ ls -l /proc
total 0
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 1
```

```
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 63
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 94
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 95
dr-xr-xr-x   4 root     users           0 Jan 31 20:37 98
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 99
-r--r--r--   1 root     root            0 Jan 31 20:37 devices
-r--r--r--   1 root     root            0 Jan 31 20:37 dma
-r--r--r--   1 root     root            0 Jan 31 20:37 filesystems
-r--r--r--   1 root     root            0 Jan 31 20:37 interrupts
-r--------   1 root     root      8654848 Jan 31 20:37 kcore
-r--r--r--   1 root     root            0 Jan 31 11:50 kmsg
-r--r--r--   1 root     root            0 Jan 31 20:37 ksyms
-r--r--r--   1 root     root            0 Jan 31 11:51 loadavg
-r--r--r--   1 root     root            0 Jan 31 20:37 meminfo
-r--r--r--   1 root     root            0 Jan 31 20:37 modules
dr-xr-xr-x   2 root     root            0 Jan 31 20:37 net
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 self
-r--r--r--   1 root     root            0 Jan 31 20:37 stat
-r--r--r--   1 root     root            0 Jan 31 20:37 uptime
-r--r--r--   1 root     root            0 Jan 31 20:37 version
$
```

(ci saranno alcuni file in più che non corrispondono a processi; l'esempio qui sopra è stato accorciato).

Notate che anche se si chiama filesystem, nessuna parte di **proc** tocca nessun disco: esiste solo nell'immaginazione del kernel. Quando si cerca di guardare una qualsiasi parte del filesystem **proc**, il kernel fa sembrare che esista da qualche parte, ma non è così; quindi, anche se c'è un file `/proc/kcore` di qualche mega, non occupa spazio disco.

4.8.3 Quale filesystem bisogna usare?

Di solito non ci sono molte ragioni per usare diversi filesystem. Al momento, l'ext2 è il più comune e probabilmente è la scelta più saggia. A seconda delle necessità globali di strutture di archiviazione, velocità, affidabilità (apparente), compatibilità e svariate altre ragioni può essere consigliabile usare un altro filesystem. La scelta va fatta caso per caso.

4.8.4 Come creare un filesystem

I filesystem vengono creati, cioè inizializzati, con il comando **mkfs**. In realtà esiste un programma separato per ciascun tipo di filesystem e **mkfs** è solo un'interfaccia che avvia quello adatto a seconda del tipo di filesystem desiderato. Il tipo viene selezionato con l'opzione **-t tipofs**.

I programmi richiamati da **mkfs** hanno interfacce su linea di comando leggermente diverse. Le opzioni più comuni ed importanti sono riassunte qui sotto; per altri dettagli consultare la pagina man.

-t tipofs
: Seleziona il tipo di filesystem

-c
: Attiva la ricerca dei blocchi danneggiati e ne inizializza di conseguenza la lista.

-l nomefile
: Legge la lista iniziale di blocchi danneggiati dal file nomefile.

Per creare un filesystem ext2 su un floppy si dovrebbero dare i seguenti comandi:

```
$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track.  Total capacity 1440 kB.
Formatting . . .  done
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ mkfs -t ext2 -l bad-blocks /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

Per prima cosa il floppy viene formattato (l'opzione **-n** fa in modo che non vengano ricercati i blocchi danneggiati); poi viene attivata la ricerca con **badblocks**, con l'output rediretto in un file, `bad-blocks`. Infine, si crea il filesystem, con l'elenco dei blocchi danneggiati letto da ciò che è stato trovato dal comando **badblocks**.

Si poteva usare nello stesso modo l'opzione **-c** di **mkfs** invece di **badblocks** ed un file separato, come riportato nell'esempio qui sotto.

```
$ mkfs -t ext2 -c /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

L'opzione **-c** è più conveniente che usare separatamente **badblocks**, ma **badblocks** è indispensabile per i controlli successivi alla creazione del filesystem.

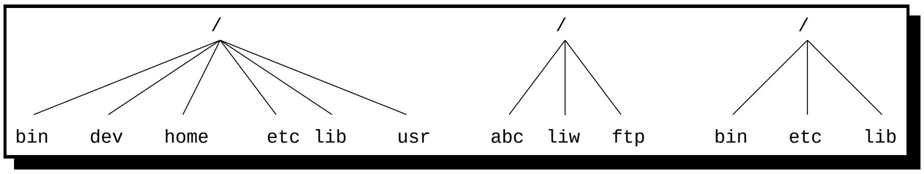
Il processo per preparare i filesystem sugli hard disk o sulle partizioni è lo stesso che per i floppy, tranne che non è necessario formattarli.

4.8.5 Montare e smontare un filesystem

Prima che si possa usare un filesystem, bisogna *montarlo*. In quel momento il sistema operativo compie varie operazioni di archiviazione in modo da essere sicuri che tutto funzioni. Dato che tutti i file in UNIX si trovano in un singolo albero di directory, l'operazione di mount farà sembrare che il nuovo filesystem sia contenuto in una sottodirectory esistente di qualche filesystem già montato.

Ad esempio, la Figura 4.3 mostra tre filesystem separati, ciascuno con la propria directory principale; quando gli ultimi due vengono montati sotto `/home` e `/usr` sul primo filesystem, abbiamo un singolo albero di directory come in Figura 4.4. I comandi per montare questi due filesystem sono riportati nel

Figura 4.3 – *Tre filesystem separati.*



seguente esempio:

```
$ mount /dev/hda2 /home
$ mount /dev/hda3 /usr
$
```

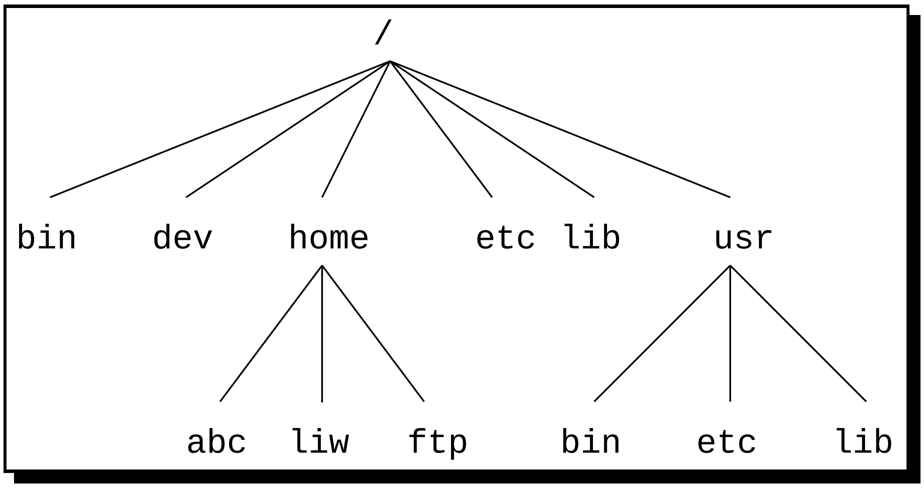
Il comando **mount** accetta due argomenti: il file di device corrispondente al disco o alla partizione che contiene il filesystem e la directory sotto cui montarlo. Dopo questi comandi il contenuto dei due filesystem sembra trovarsi rispettivamente nelle directory `/home` e `/usr`; si può dire a questo punto che "`/dev/hda2` *è montato sotto* `/home`" e la stessa cosa per `/usr`. Per vedere i due filesystem basta guardare nelle directory su cui sono stati montati, come se fossero una qualsiasi altra directory. Notate la differenza tra il file di device, `/dev/hda2`, e la directory su cui viene montato, `/home`: il file di device dà accesso ai contenuti fisici del disco, la directory su cui viene montato ai file che vi si trovano. La directory su cui viene montato un filesystem si chiama *punto di mount*.

Linux supporta molti tipi di filesystem; **mount** cerca di indovinare il tipo di quello che si sta montando. Si può anche usare l'opzione **-t tipofs** per specificare direttamente il tipo: talvolta è necessaria, dato che l'euristica usata da **mount** non sempre funziona. Ad esempio per montare un floppy MS-DOS si usa il seguente comando:

```
$ mount -t msdos /dev/fd0 /floppy
$
```

La directory su cui si monta il filesystem può anche non essere vuota, ma deve esistere. Qualsiasi file vi si trovi non potrà essere richiamato per nome mentre

Figura 4.4 – *Sono stati montati* `/home` *e* `/usr`*.*



il filesystem è montato (qualsiasi file già aperto sarà ancora accessibile; i file a cui puntano hard link da altre directory possono essere accessibili usando quei nomi). Questo procedimento permette di non perdere nessun dato e può anche essere utile: ad esempio, alcuni vogliono avere `/tmp` e `/var/tmp` sinonimi, e rendere `/tmp` un link simbolico a `/var/tmp`.

Quando si avvia il filesystem, prima che venga montato `/var`, viene usata una directory `/var/tmp` che risiede sul filesystem radice; quando viene montato `/var`, la directory `/var/tmp` sul filesystem radice diventa inaccessibile. Se non esistesse `/var/tmp`, sarebbe impossibile usare i file temporanei prima di montare `/var`.

Se non volete scrivere niente su un filesystem, usate l'opzione **`-r`** per montarlo *a sola lettura*; in questo modo il kernel interromperà qualsiasi tentativo di scrivervi ed eviterà di aggiornare i tempi di accesso ai file negli inode. È indispensabile usare i mount a sola lettura per i mezzi non scrivibili, come i CD-ROM.

Il lettore attento avrà già notato un piccolo problema logistico: come viene montato il primo filesystem (chiamato *filesystem radice*, perché contiene la directory principale), dato che ovviamente non può essere montato su un altro filesystem? Beh, la risposta è che si usa la magia[7]. Il filesystem radice viene montato magicamente al momento dell'avvio del sistema e resta sempre montato; se all'avvio non è possibile farlo, il sistema non parte. Il nome del filesystem che viene montato magicamente come radice è compilato nel kernel, o impostato usando LILO o **rdev**.

Il filesystem radice viene di solito montato inizialmente a sola lettura, gli script di avvio poi attivano **fsck** per verificare che sia valido e se non ci sono problemi lo rimonteranno con i permessi di scrittura. **fsck** non deve essere usato su un filesystem montato, dato che qualsiasi modifica al filesystem mentre sta girando **fsck** *farà* dei danni. Dato che il filesystem radice è montato a sola lettura mentre viene controllato, **fsck** può correggere qualsiasi problema senza preoccupazioni, dato che l'operazione di remount ripulirà tutti i metadati che il filesystem tiene in memoria.

Su molti sistemi ci sono altri filesystem che devono essere montati automaticamente al momento dell'avvio: questi vengono specificati nel file `/etc/fstab`; consultate la pagina man di `fstab` per avere dettagli sul suo formato. I dettagli di quando esattamente vengono montati i vari filesystem dipendono da molti fattori, e possono essere configurati, se serve, dall'amministratore; vedere il Capitolo 6.

Quando non serve più che un filesystem sia montato, può essere smontato usando **umount**[8]. **umount** accetta un argomento: il file di device o il punto di mount. Ad esempio, per smontare le directory dell'esempio precedente, si possono usare i comandi

```
$ umount /dev/hda2
$ umount /usr
$
```

Consultare la pagina man per avere altre istruzioni su come usare il comando. È indispensabile smontare sempre i floppy montati: *non li tirate direttamente fuori dal lettore!* A causa delle operazioni di cache su disco, i dati non vengono necessariamente scritti sul floppy finché non lo smontate, quindi rimuovendo

[7] Per altre informazioni consultate i sorgenti del kernel o la *Kernel Hackers' Guide* [5].

[8] In effetti dovrebbe essere **unmount**, ma la 'n' è scomparsa misteriosamente negli anni '70, e non è stata più vista da allora. Per favore, se la trovate, riconsegnatela ai Bell Labs, NJ.

il floppy dal lettore troppo presto se ne possono corrompere i contenuti. Se leggete soltanto dal floppy non è molto probabile, ma se ci scrivete, anche accidentalmente, i risultati possono essere catastrofici.

Montare e smontare un filesystem richiede privilegi di superutente, cioè lo può fare solo root; la ragione è che se chiunque potesse montare un dischetto su una qualsiasi directory, sarebbe piuttosto facile creare un floppy diciamo con un cavallo di Troia camuffato da `/bin/sh`, o da un qualsiasi programma usato spesso. Comunque, spesso è necessario permettere agli utenti di usare i floppy, ed ecco alcuni modi per poterlo fare:

- Dare agli utenti la password di root. Ovviamente è un buco di sicurezza, ma è la soluzione più facile ed adatta a quei sistemi dove non c'è comunque bisogno di sicurezza, come molti sistemi personali non in rete.

- Usare un programma come **sudo** per permettere agli utenti di usare **mount**. Anche questo crea problemi di sicurezza, ma non dà direttamente privilegi di superutente a chiunque[9].

- Fare usare agli utenti **mtools**, un pacchetto per manipolare i filesystem MS-DOS senza montarli. Funziona bene se l'unica cosa che serve sono dischetti MS-DOS, altrimenti è piuttosto complicato.

- Elencare i dispositivi floppy e i loro punti di mount permessi, insieme con le opzioni adatte, in `/etc/fstab`.

L'ultima alternativa può essere implementata aggiungendo una linea come la seguente al file `/etc/fstab`:

```
/dev/fd0    /floppy      msdos   user,noauto   0  0
```

Le colonne sono: file di device da montare, directory su cui montarlo, tipo di filesystem, opzioni, frequenza di backup (usata da **dump**) e numero di passaggio di **fsck** (per specificare l'ordine in cui controllare i filesystem all'avvio: con 0 non vengono controllati).

L'opzione **noauto** evita che il filesystem venga montato automaticamente all'avvio del sistema (cioè non lo fa montare da **mount -a**). L'opzione **user** permette a qualsiasi utente di montare il filesystem e, per sicurezza, disabilita

[9] Richiede però qualche secondo di concentrazione da parte degli utenti.

l'esecuzione di programmi (normali o setuid) e l'interpretazione di file di device dal filesystem montato. Fatto questo, qualsiasi utente può montare un floppy con un filesystem msdos usando il seguente comando:

```
$ mount /floppy
$
```

Il floppy può (e deve, naturalmente) essere smontato con il corrispondente comando **umount**.

Se volete dare accesso a diversi tipi di floppy, dovete dare diversi punti di mount. Le impostazioni possono essere diverse per ogni punto di mount, ad esempio, per dare accesso sia a floppy MS-DOS che ext2, si possono inserire in `/etc/fstab`:

```
/dev/fd0    /dosfloppy    msdos   user,noauto  0  0
/dev/fd0    /ext2floppy   ext2    user,noauto  0  0
```

Probabilmente vorrete restringere l'accesso ai filesystem MS-DOS (non solo per il floppy) usando le opzioni **uid**, **gid**, ed **umask**, che sono descritte in dettaglio nella pagina man di **mount**. Se non state attenti, montare un filesystem MS-DOS dà a tutti almeno il permesso di lettura ai file che vi si trovano, il che non è una buona idea.

4.8.6 Il controllo dell'integrità di un filesystem con fsck

I filesystem sono creature complesse e come tali tendono ad essere in qualche modo propense ad avere errori. Si possono controllare la correttezza e la validità di un filesystem usando il comando **fsck**, che può essere usato per riparare i piccoli errori che trova e per avvisare l'utente se ci sono problemi irreparabili. Fortunatamente al codice per implementare i filesystem viene fatto un debug piuttosto efficace, quindi in genere non ci sono problemi e se ci sono sono causati da mancanza di corrente, guasti hardware o errori dell'operatore, ad esempio non fare correttamente lo shutdown del sistema.

La maggior parte dei filesystem fanno **fsck** automaticamente all'avvio del sistema, in modo che gli errori vengono individuati (e, speriamo, corretti) prima che il filesystem venga usato. Usare un filesystem corrotto tende a peggiorare le cose: se le strutture di dati non sono integre, usare il filesystem probabilmente le danneggerà ancora di più e verranno persi sempre più dati. Comunque, **fsck** può metterci un po' di tempo per controllare un filesystem grande e, dato che non ci sono quasi mai errori se il sistema viene spento in modo corretto, si possono usare un paio di trucchi per evitare di fare il controllo in casi del genere: il primo è che se esiste il file `/etc/fastboot` non vengono fatti controlli. Il secondo è che il filesystem ext2 ha un codice speciale nel superblocco che dice se il filesystem è stato smontato in maniera corretta l'ultima volta; in questo modo **e2fsck** (la versione di **fsck** per filesystem ext2) può evitare di controllare i filesystem se il codice indica che è stato smontato correttamente (l'assunzione è che se il filesystem è stato smontato bene non ci sono problemi). La validità del trucco del file `/etc/fastboot` dipende dallo script di avvio, ma il trucco dell'ext2 funziona tutte le volte che si usa **e2fsck**—per evitarlo deve essere esplicitamente indicata un'opzione (consultate la pagina man di **e2fsck** per i dettagli).

Il controllo automatico funziona solo per i filesystem che vengono montati automaticamente all'avvio; usate **fsck** a mano per gli altri, come ad esempio per i floppy.

Se **fsck** trova dei problemi irrecuperabili dovete avere una conoscenza profonda di come funzionano i filesystem in generale e del tipo del filesystem corrotto in particolare, oppure dovete avere dei buoni backup. La seconda possibilità è più facile (anche se spesso più noiosa), per la prima ci si può appoggiare ad un amico, ai newsgroup ed alle mailing list su Linux o ad una qualche altra forma di aiuto, se non avete voi stessi le nozioni necessarie. Mi piacerebbe dirvi di più sull'argomento, ma mi trattiene la mia mancanza di esperienza e di conoscenza in questo campo. Vi dovrebbe essere utile il programma **debugfs** di Theodore T'so.

fsck deve essere usato solo sui filesystem non montati, mai su quelli montati (con l'eccezione della radice a sola lettura durante l'avvio), perché accede al disco a basso livello e quindi può modificare il filesystem senza che il sistema operativo se ne accorga. È *veramente* un problema, se si fa confondere il sistema operativo.

4.8.7 Il controllo degli errori sul disco con badblocks

Può essere una buona idea controllare periodicamente se ci sono blocchi danneggiati, con il comando **badblocks**, che dà in output un elenco dei numeri dei blocchi danneggiati che trova. Si può usare questa lista per farla registrare nelle strutture dati del filesystem a **fsck** in modo che il sistema operativo non provi ad usare i blocchi danneggiati per scriverci dei dati. Nell'esempio viene spiegato come fare:

```
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ fsck -t ext2 -l bad-blocks /dev/fd0H1440
Parallelizing fsck version 0.5a (5-Apr-94)
e2fsck 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.

/dev/fd0H1440: ***** FILE SYSTEM WAS MODIFIED *****
/dev/fd0H1440: 11/360 files, 63/1440 blocks
$
```

Se **badblocks** riporta un blocco che già è stato usato, **e2fsck** prova a spostare il blocco in un altro punto. Se il blocco è già stato veramente danneggiato e l'errore non era solo marginale, i contenuti del file possono già essere corrotti.

4.8.8 Combattere la frammentazione

Quando si scrive un file sul disco, non lo si può sempre fare su blocchi consecutivi; un file per cui non è stato possibile farlo si definisce *frammentato*. Ci vuole più tempo per leggere un file frammentato, dato che la testina di lettura e scrittura del disco si deve spostare di più, quindi è preferibile evitare la frammentazione, anche se questa è un problema meno grave nei sistemi con una buona cache di buffer con il read-ahead.

Il filesystem ext2 tenta di mantenere la frammentazione al minimo, tenendo tutti i blocchi di un file vicini, anche se non possono essere immagazzinati in settori consecutivi. L'ext2 in effetti alloca sempre il blocco libero che si trova più vicino agli altri blocchi di un file. Per l'ext2, quindi, è molto raro doversi preoccupare della frammentazione, ma esiste un programma di deframmentazione anche per i filesystem ext2: consultate [9].

Ci sono molti programmi di deframmentazione per MS-DOS che spostano i blocchi nel filesystem per rimuovere la deframmentazione. Per altri tipi di filesystem si deve fare un backup completo, ricreare il filesystem e recuperare i dati dai backup. Fare il backup prima di deframmentare un filesystem è una buona idea in ogni caso, dato che molte cose possono andare storte nel processo.

4.8.9 Altri strumenti per tutti i filesystem

Anche altri strumenti sono utili per gestire i filesystem: **df** mostra lo spazio disco libero su uno o più filesystem, **du** quello occupato da una directory e da tutti i suoi file. Si possono usare entrambi questi comandi per ricercare chi spreca spazio disco.

sync forza la scrittura sul disco di tutti i blocchi non scritti nella cache di buffer (vedi la Sezione *La cache di buffer* nel Capitolo 5). Normalmente non c'è bisogno di farlo a mano: lo fa automaticamente il processo daemon **update**. Può essere utile in caso di catastrofe, ad esempio se **update** o il suo aiutante **bdflush** muoiono, o se dovete interrompere la corrente *ora* e non potete aspettare che venga avviato **update**.

4.8.10 Altri strumenti per il filesystem ext2

In aggiunta al comando per creare i filesystem (**mke2fs**) e quello per controllarli (**e2fsck**), accessibili direttamente o attraverso le interfacce indipendenti dal tipo di filesystem, il filesystem ext2 ha altri strumenti utili.

tune2fs migliora i parametri del filesystem. Alcuni di quelli più interessanti sono:

- Il numero massimo di mount. **e2fsck** forza un controllo ogni volta che un filesystem viene montato troppe volte, anche se ha il codice di filesystem integro. Per un sistema che viene usato per sviluppare o testare il sistema, può essere una buona idea ridurre questo limite.

- Il tempo massimo tra i controlli. **e2fsck** può anche forzare un controllo dopo un tempo massimo, anche se c'è il codice di filesystem pulito e il filesystem non è stato montato molto spesso. Questo parametro può essere anche disabilitato.

- Il numero di blocchi riservato per root. L'ext2 riserva alcuni blocchi per root in modo che se il filesystem si riempie sia ancora possibile fare dell'amministrazione di sistema senza dover cancellare niente. La percentuale riservata è per default del 5%, che sulla maggior parte dei dischi non è tanto da essere uno spreco. Comunque, per i floppy non c'è motivo per riservare niente.

Consultate la pagina man di **tune2fs** per altre informazioni.

dumpe2fs dà delle informazioni su un filesystem ext2, prese per la maggior parte dal suo superblocco. In Figura 4.5 viene mostrato un esempio di output. Alcune delle informazioni dell'output sono tecniche e richiedono una comprensione di come funziona il filesystem (vedere [10]), ma la peggior parte è facilmente comprensibile.

debugfs è un debugger per i filesystem: permette accesso diretto alle strutture dati immagazzinate sul disco e quindi può essere usato per riparare gli errori che non possono essere corretti automaticamente con **fsck**. È stato anche usato per recuperare dei file cancellati; comunque, **debugfs** richiede di capire a fondo quello che si sta facendo: se non riuscite a capire qualcosa potreste distruggere tutti i dati.

Per fare il backup di un filesystem ext2 si possono usare **dump** e **restore**: si tratta di versioni specifiche per questo tipo di filesystem degli strumenti tradizionali di backup di UNIX. Vedere il Capitolo 10 per altre informazioni sui backup.

4.9 Dischi senza filesystem

Non tutti i dischi o le partizioni vengono usati come filesystem: ad esempio le partizioni di swap non hanno filesystem. Molti floppy vengono usati come fossero dei nastri, scrivendo direttamente un **tar** o un altro file sul disco vuoto, senza un filesystem.

Evitare il filesystem ha il vantaggio di rendere utilizzabile una parte maggiore del disco, dato che questo occupa sempre dello spazio per l'archiviazione; rende anche il disco più facilmente compatibile con gli altri sistemi: ad esempio, il formato dei file **tar** è lo stesso su tutti i sistemi, mentre i filesystem sono diversi. Vi abituerete presto ai dischi senza filesystem se ne avrete bisogno. Neanche i floppy di boot di Linux hanno necessariamente un filesystem, anche se possono averlo.

Una ragione di usare i dischi senza filesystem è che se ne possono fare copie identiche. Ad esempio, se il disco contiene un filesystem parzialmente danneggiato, è una buona idea farne una copia esatta prima di tentare di ri-

Figura 4.5 – *Esempio di output di* ***dumpe2fs****.*

```
dumpe2fs 0.5b, 11-Mar-95 for EXT2 FS 0.5a, 94/10/23
Filesystem magic number:  0xEF53
Filesystem state:         clean
Errors behavior:          Continue
Inode count:              360
Block count:              1440
Reserved block count:     72
Free blocks:              1133
Free inodes:              326
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         360
Last mount time:          Tue Aug  8 01:52:52 1995
Last write time:          Tue Aug  8 01:53:28 1995
Mount count:              3
Maximum mount count:      20
Last checked:             Tue Aug  8 01:06:31 1995
Check interval:           0
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)

Group 0:
  Block bitmap at 3, Inode bitmap at 4, Inode table at 5
  1133 free blocks, 326 free inodes, 2 directories
  Free blocks: 307-1439
  Free inodes: 35-360
```

pararlo, in modo da poter ricominciare da capo se cercando di aggiustarlo lo si danneggia ancora di più. Un modo di farlo è usare **dd**:

```
$ dd if=/dev/fd0H1440 of=floppy-image
2880+0 records in
2880+0 records out
$ dd if=floppy-image of=/dev/fd0H1440
2880+0 records in
2880+0 records out
$
```

Il primo comando **dd** crea un'immagine esatta del floppy nel file `floppy-image`, il secondo copia l'immagine nel floppy (presumibilmente l'utente ha cambiato floppy prima del secondo comando, altrimenti la coppia di comandi non è molto utile).

4.10 Allocare spazio disco

4.10.1 Schemi di partizionamento

Non è facile partizionare un disco nel modo migliore possibile. Peggio ancora, non c'è nessun modo corretto per farlo, sono coinvolti troppi fattori.

La procedura più comune da seguire è avere un filesystem radice (relativamente) piccolo, che contenga `/bin`, `/etc`, `/dev`, `/lib`, `/tmp` ed altre cose necessarie per fare partire il sistema: in questo modo il filesystem radice (nella sua partizione o sul suo disco) è tutto quello che serve per avviare il sistema. Il ragionamento è che se il filesystem radice è piccolo e non viene usato pesantemente, è meno probabile che venga corrotto quando si ha un crash del sistema e quindi sarà più facile risolvere qualsiasi problema che si venga a creare. Si creano poi delle partizioni separate o si usano dischi separati per l'albero delle directory sotto `/usr`, le home directory degli utenti (di solito sotto `/home`) e lo spazio di swap. Separare le home directory (con i file degli utenti) in una partizione distinta rende più facili i backup, dato che spesso non è necessario farlo per i programmi (che risiedono sotto `/usr`). In un ambiente di rete è anche possibile condividere `/usr` tra svariate macchine (ad esempio usando NFS) e riducendo così lo spazio disco totale richiesto di diverse decine o centinaia di megabyte moltiplicato per il numero di macchine.

Il problema di avere molte partizioni è che ciò divide lo spazio libero totale in piccoli pezzi; oggigiorno che i dischi e (speriamo) i sistemi operativi sono più affidabili, molte persone preferiscono avere solo una partizione che contenga tutti i file. D'altra parte può essere meno doloroso fare il backup (e il restore) di una partizione piccola.

Per un hard disk piccolo (assumendo che non sviluppiate il kernel), il modo migliore è probabilmente avere una sola partizione. Per dischi grandi è probabilmente meglio avere alcune partizioni grandi, giusto in caso vada bene qualcosa (notate che 'piccola' o 'grande' sono termini usati in senso relativo: le soglie sono dettate dal vostro bisogno di spazio disco).

Se avete più di un disco potete pensare di tenere il filesystem radice (compreso `/usr`) su uno e le home degli utenti su un altro.

È una buona idea essere pronti a sperimentare un po' con diversi schemi di partizionamento (nel tempo, non solo alla prima installazione del sistema): è un lavoraccio, dato che richiede essenzialmente di reinstallare il sistema da zero diverse volte, ma è l'unico modo di essere sicuri di farlo bene.

4.10.2 Requisiti di spazio

Nella documentazione della vostra distribuzione di Linux troverete alcune indicazioni sullo spazio disco necessario per svariate configurazioni; i programmi installati separatamente possono fare lo stesso. In questo modo vi sarà possibile pianificare l'uso dello spazio disco, ma dovreste prepararvi per il futuro e riservare dello spazio extra per le cose di cui vi accorgerete di avere bisogno.

La quantità di spazio che vi serve per i file degli utenti dipende da cosa essi vogliono fare. Sembra che la maggior parte delle persone abbiano bisogno del maggiore spazio possibile per i propri file, ma la quantità con cui vivranno in pace varia molto: alcuni lavorano solo sui testi e sopravviveranno con qualche megabyte, altri fanno manipolazione pesante di immagini e avranno bisogno di gigabyte.

Tra l'altro, quando si paragonano le dimensioni dei file in kilobyte o in megabyte e lo spazio disco misurato in megabyte, è importante sapere che le due unità possono essere diverse. Alcuni produttori di dischi amano pensare che un kilobyte è 1000 byte e che un megabyte è 1000 kilobyte, mentre tutto il resto del mondo informatico usa 1024 per entrambi i fattori. Quindi, il mio hard disk da 345 MB in realtà è un hard disk da 330 MB[10].

L'allocazione dello spazio di swap viene discussa nella Sezione *Allocare lo spazio di swap* nel Capitolo 5.

4.10.3 Esempi di allocazione di hard disk

Io avevo un hard disk da 109 MB; ora ne uso uno da 330 MB. Spiegherò come e perché ho partizionato entrambi.

Ho partizionato il disco da 109 MB in moltissimi modi, a seconda delle mie necessità e dei sistemi operativi che usavo: illustrerò due scenari tipici. All'inizio usavo l'MS-DOS insieme a Linux; per quello avevo bisogno più o meno di 20 MB di hard disk, o lo spazio appena sufficiente per avere l'MS-DOS, un compilatore C, un editor, pochi altri programmi, quello su cui stavo lavorando e un minimo per non farmi sentire claustrofobico. Per Linux avevo una partizione di swap di 10 MB ed il resto, cioè 79 MB, era una partizione singola con tutti i file che avevo sotto Linux. Ho sperimentato con delle partizioni radice, `/usr` e `/home` separate, ma non c'era mai abbastanza spazio disco libero insieme per fare molto di interessante.

Quando non ho più avuto bisogno dell'MS-DOS, ho ripartizionato l'hard disk in modo di avere una partizione di swap di 12 MB ed il resto di nuovo in un singolo filesystem.

Il disco da 330 MB è partizionato in diverse partizioni, così:

5 MB	filesystem radice
10 MB	partizione di swap
180 MB	filesystem `/usr`
120 MB	filesystem `/home`
15 MB	partizione di scratch

La partizione di scratch è per giocherellare con cose che richiedono una propria partizione, cioè provare diverse distribuzioni di Linux, o paragonare le velocità dei filesystem. Quando non serve per nient'altro, la uso come spazio di swap (mi piace tenere moltissime finestre aperte).

[10] *Sic transit discus mundi.*

4.10.4 Aggiungere altro spazio disco per Linux

Aggiungere altro spazio disco per Linux è facile, almeno dopo che l'hardware è stato installato correttamente (cosa che va oltre lo scopo di questo libro): si formatta, se necessario, poi si creano le partizioni ed i filesystem come è stato spiegato nei paragrafi precedenti e si aggiungono le linee adatte in `/etc/fstab` in modo che vengano montate automaticamente.

4.10.5 Suggerimenti per risparmiare spazio su disco

Il modo migliore per risparmiare spazio disco è evitare di installare programmi non necessari. La maggior parte delle distribuzioni hanno un'opzione per installare solo una parte dei pacchetti che contengono e analizzando le vostre necessità potrete accorgervi di non aver bisogno della maggior parte di essi. Vi aiuterà a risparmiare spazio disco, dato che molti programmi sono piuttosto grandi. Anche se avete bisogno di un pacchetto o di un programma particolare, non è detto che vi servano tutti. Ad esempio può non essere necessaria parte della documentazione in linea, o alcuni file Elisp di GNU Emacs, font di X11 o librerie di programmazione.

Se non potete disinstallare i pacchetti, potete provare con la compressione. I programmi di compressione come **gzip** o **zip** comprimono (e decomprimono) file singoli o gruppi di essi. Il sistema **gzexe** comprime e decomprime i programmi in maniera invisibile all'utente (i programmi che non vengono usati vengono compressi e decompressi quando si usano), il sistema sperimentale **DouBle** comprime tutti i file di un filesystem in maniera invisibile ai programmi che li usano (se avete familiarità con prodotti come Stacker per l'MS-DOS, il principio è lo stesso).

Capitolo 5

La gestione della memoria

> *Minnet, jag har tappat mitt minne, är jag svensk eller finne, kommer inte ihåg...*
>
> (Bosse Österberg)

Questa sezione descrive le caratteristiche di gestione della memoria di Linux, cioè la memoria virtuale e la cache di buffer del disco. Ne vengono descritti lo scopo, il funzionamento e le azioni necessarie da parte dell'amministratore di sistema.

5.1 Cosa è la memoria virtuale?

Linux supporta la *memoria virtuale*, cioè l'utilizzo di un disco come estensione della RAM in modo da aumentare di conseguenza la dimensione della memoria utilizzabile. Il kernel scrive il contenuto di un blocco di memoria che al momento non viene utilizzato sull'hard disk, in modo che la memoria reale possa essere usata per altre cose, e quando il contenuto originale serve ancora viene spostato di nuovo nella RAM. Questo processo è completamente trasparente all'utente: i programmi che girano sotto Linux vedono soltanto una memoria disponibile più grande del vero e non sanno che parte di essa risiede sull'hard disk. Naturalmente leggere e scrivere sull'hard disk è più lento (dell'ordine di un migliaio di volte) che usare la memoria vera, quindi i programmi non sono ugualmente veloci. La parte dell'hard disk che viene usato come memoria virtuale si chiama *spazio di swap*.

Linux può usare come spazio di swap sia un file normale nel filesystem che una partizione separata; una partizione è più veloce, ma è più facile modificare la dimensione di un file (non c'è bisogno di ripartizionare l'intero hard disk e probabilmente installare tutto da zero). Quando sapete la quantità di spazio di swap di cui avete bisogno vi conviene usare una partizione, ma se non siete sicuri create un file, usate il sistema per un po' per farvi un'idea e poi fate una partizione quando siete sicuri sulla sua dimensione.

Dovreste anche sapere che Linux permette di usare diversi file e partizioni di swap nello stesso momento. Per questo, se vi serve per un periodo uno spazio di swap più grande del normale, potete impostare un file temporaneo, invece di tenerlo tutto sempre allocato.

Una nota sulla terminologia dei sistemi operativi: in informatica di solito si distingue tra lo swapping (scrivere tutto il processo nello spazio di swap) e il paging (scriverci solo delle parti di dimensione fissa, di solito alcuni kilobyte, alla volta). Il paging di solito è più efficiente, ed è quello che fa Linux, ma la terminologia tradizionale di Linux parla comunque di swapping[1].

5.2 La creazione di uno spazio di swap

Un file di swap è un file ordinario e non viene considerato in maniera speciale dal kernel, l'unica cosa che importa al kernel è che non deve avere buchi e che sia preparato all'uso con **mkswap**. Deve risiedere su un disco locale, comunque, e non può trovarsi su un filesystem montato via NFS per ragioni di implementazione.

La parte sui buchi è importante: il file riserva lo spazio disco in modo che il kernel possa fare velocemente lo swap di una pagina senza dover fare tutti i passi necessari per allocare un settore di un disco ad un file. Il kernel usa semplicemente i settori che sono stati allocati per il file; dato che un buco significa che per quel posto nel file non sono allocati settori, non è bene che il kernel provi ad usarli.

[1] Facendo quindi innervosire in modo spaventoso moltissimi informatici.

Un modo buono per creare il file di swap senza buchi è usare questo comando:

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
1024+0 records in
1024+0 records out
$
```

dove `/extra-swap` è il nome del file di swap e la sua dimensione viene data dopo `count=`. È meglio che la dimensione sia un multiplo di 4, perché il kernel manda in swap *pagine di memoria* di 4 kilobyte, quindi se lo spazio non è multiplo di 4 Kb una parte può restare inutilizzata.

Anche una partizione di swap non è in nessun modo una partizione speciale: la si crea come una qualsiasi altra partizione, l'unica differenza è che viene usata così com'è e non contiene un filesystem. È una buona idea segnare le partizioni di swap come tipo 82 (Linux swap); in questo modo renderete più chiara la tabella delle partizioni, anche se non è strettamente necessario per il kernel.

Dopo aver creato un file o una partizione di swap bisogna scrivere al suo inizio una firma che contiene delle informazioni di amministrazione usate dal kernel. Il comando per farlo è **mkswap**, che si usa così:

```
$ mkswap /extra-swap 1024
Setting up swapspace, size = 1044480 bytes
$
```

Notate che lo spazio di swap non è ancora in uso: esiste, ma il kernel non lo usa per fornire memoria virtuale.

Va fatta molta attenzione nell'usare **mkswap**, dato che questo non controlla che il file o la partizione non siano usati per altre cose. *Potete facilmente sovrascrivere file e partizioni importanti!* Per fortuna di solito si usa **mkswap** solo quando si installa il sistema.

Il gestore di memoria di Linux limita la dimensione di ciascuno spazio di swap a circa 127 MB (per varie ragioni tecniche, il limite reale è $(4096 - 10) \cdot 8 \cdot 4096 = 133890048$ byte, o 127.6875 megabyte). Potete comunque usare fino a 16 spazi di swap simultaneamente, per un totale di circa 2 GB[2].

[2] Un gigabyte qui, un gigabyte là, presto cominceremo a parlare di memoria vera.

5.2.1 Come si usa lo spazio di swap

Uno spazio di swap inizializzato si mette in uso con **swapon**, che comunica al kernel che può essere utilizzato. Come argomento viene dato il percorso per il file o la partizione, quindi per cominciare ad usare uno spazio di swap temporaneo si può fare:

```
$ swapon /extra-swap
$
```

Gli spazi di swap possono essere usati automaticamente elencandoli nel file `/etc/fstab`.

```
/dev/hda8        none        swap        sw      0      0
/swapfile        none        swap        sw      0      0
```

Gli script di avvio fanno partire il comando **swapon -a**, che comincerà ad usare come swap tutti gli spazi elencati in `/etc/fstab`. Il comando **swapon** è quindi necessario solo se si usano spazi di swap supplementari.

Si può controllare l'utilizzo degli spazi di swap usando il comando **free**, che dice la quantità totale di spazio di swap usata.

```
$ free
             total       used       free     shared    buffers
Mem:         15152      14896        256      12404       2528
-/+ buffers:            12368       2784
Swap:        32452       6684      25768
$
```

La prima linea di output (`Mem:`) mostra la memoria fisica. La colonna `total` non considera la memoria fisica usata dal kernel, che in genere è circa un megabyte, `used` mostra la quantità di memoria usata (nella seconda linea non vengono contati i buffer), `free` quella totalmente inutilizzata e `shared` quella condivisa da diversi processi: più è, meglio è. La colonna `buffer` mostra la dimensione corrente della cache di buffer del disco.

L'ultima linea (`Swap:`) mostra le stesse informazioni per gli spazi di swap. Se questa linea contiene tutti zeri, non avete attivato lo spazio di swap.

Le stesse informazioni sono disponibili con **top** o usando il filesystem proc, in particolare il file `/proc/meminfo`. Al momento è difficile avere delle informazioni sull'uso di uno spazio di swap specifico.

Uno spazio di swap può essere disabilitato con **swapoff**; in genere non è necessario farlo, tranne che per gli spazi di swap temporanei. Le pagine in uso nello spazio di swap vengono per prima cosa copiate nella memoria; se non c'è memoria fisica sufficiente vengono messe in un altro spazio di swap. Se non c'è abbastanza memoria virtuale per mantenere tutte le pagine Linux comincerà a fare rumore; dopo un po' di tempo dovrebbe tornare normale, ma nel frattempo il sistema è inutilizzabile. Bisognerebbe sempre controllare (ad esempio con **free**) che ci sia abbastanza memoria libera prima di disabilitare uno spazio di swap.

Tutti gli spazi di swap che vengono usati automaticamente con **swapon -a** possono essere disabilitati usando **swapoff -a**, che va a guardare in `/etc/fstab` per vedere quali spazi rimuovere. Gli spazi di swap attivati manualmente rimarranno in uso.

Talvolta può venire usato spazio di swap anche se c'è molta memoria fisica libera; ad esempio se ad un certo punto c'è bisogno di fare swap, ma poi un processo grande che occupava molta memoria fisica termina e libera la memoria. I dati messi in swap non vengono reinseriti in memoria automaticamente, ma solo quando servono, quindi la memoria fisica può restare libera per parecchio tempo. Non c'è bisogno di preoccuparsene, ma può essere confortante sapere cosa sta succedendo.

5.3 La condivisione dello spazio di swap con altri sistemi operativi

La memoria virtuale viene usata da molti sistemi operativi; dato che ne hanno bisogno solo mentre stanno funzionando, cioè mai allo stesso tempo, gli spazi di swap di tutti i sistemi operativi presenti meno uno vanno sprecati, e sarebbe più efficiente se condividessero un solo spazio di swap. È possibile, ma richiede un po' di impegno. Il Tips-HOWTO [11] contiene dei suggerimenti su come fare.

5.4 Allocare lo spazio di swap

Alcuni vi diranno di allocare uno spazio di swap corrispondente al doppio della vostra memoria fisica, ma è una regola approssimativa. Ecco come fare in modo corretto:

1. Stimate di quanta memoria avete bisogno: calcolate la quantità più alta di memoria di cui avrete mai bisogno in una volta sola, cioè la somma del fabbisogno di tutti i programmi che volete far girare contemporaneamente: lo potete fare facendoli girare tutti insieme.

   Ad esempio, se volete usare X dovreste allocargli circa 8 MB, gcc vuole qualche megabyte (alcuni file hanno bisogno di moltissima memoria, fino a decine di megabyte, ma di solito quattro bastano) e così via. Il kernel userà per sé circa un megabyte, le normali shell ed altri programmi qualche centinaia di kilobyte (diciamo un megabyte tutti insieme). Non c'è bisogno di tentare di essere esatti, basta una stima rozza, magari per eccesso.

   Ricordatevi che se ci sono diverse persone che usano il sistema in contemporanea consumeranno tutti della memoria; comunque, se due persone usano lo stesso programma contemporaneamente l'utilizzo complessivo di memoria non è di solito doppio, dato che le pagine di codice e le librerie condivise vengono usate solo una volta.

   Per stimare le necessità di memoria sono utili i comandi **free** e **ps**.

2. Aggiungete per sicurezza un po' di memoria alla stima del passo 1: la dimensione di alcuni programmi sarà sbagliata, o forse vi dimenticherete alcuni programmi che volete usare ed è meglio avere un po' di spazio extra per qualsiasi evenienza. Un paio di megabyte dovrebbero andare bene (è meglio allocare troppo spazio di swap che troppo poco, ma non c'è bisogno di strafare e allocare l'intero disco, dato che lo spazio di swap inutilizzato è sprecato; vedi sopra come fare per aggiungere altro spazio). Inoltre, dato che si lavora meglio con le cifre tonde, arrotondate il valore al megabyte superiore.

3. Basandovi sul calcolo qui sopra sapete di quanta memoria avete bisogno in totale. Per allocare lo spazio di swap dovete dunque sottrarre la dimensione della memoria fisica dalla memoria totale (in alcune versioni di UNIX dovete allocare dello spazio di swap anche per un'immagine della memoria fisica, quindi il totale calcolato nel passo 2 è lo swap necessario e non dovete fare la sottrazione).

4. Se lo spazio di swap calcolato è molto più grande della memoria fisica (più del doppio) dovreste investire in più memoria fisica, o le prestazioni saranno troppo basse.

È una buona idea avere comunque dello spazio di swap, anche se i calcoli indicano che non ne avete bisogno; Linux lo usa in modo piuttosto aggressivo, in modo che possa essere tenuta libera più memoria fisica possibile, e manderà in swap delle pagine di memoria che non vengono usate anche se non c'è ancora bisogno di memoria per altre cose. Un comportamento del genere evita di aspettare di fare swap nel momento in cui ce n'è bisogno—si può fare prima, quando il disco altrimenti non sta lavorando. Lo spazio di swap può essere suddiviso tra vari dischi, in modo da poter in qualche modo migliorare le prestazioni, a seconda delle loro velocità e degli schemi di accesso. Potete fare degli esperimenti con schemi diversi, ma farlo in modo corretto non è semplice. Non credete a chi dice che uno schema è sicuramente superiore ad un altro, perché non sempre è vero.

5.5 La cache di buffer

Leggere da un disco[3] è molto più lento che accedere a della memoria (reale). Oltre a ciò, è comune leggere la stessa parte del disco diverse volte in brevi periodi di tempo. Ad esempio, si può prima leggere un messaggio di posta elettronica, poi caricare lo stesso messaggio in un editor per rispondere, poi farlo leggere al programma di posta per copiarlo in una cartella. Oppure, considerate quanto spesso si può usare il comando **ls** su un sistema con molti utenti. Leggendo le informazioni dal disco e trattenendole in memoria finché non servono più si possono velocizzare le letture successive alla prima. Questo processo si chiama *bufferizzazione del disco*, e la memoria usata allo scopo è la *cache di buffer*.

Dato che la memoria è, sfortunatamente, una risorsa finita, anzi, scarsa, la cache di buffer di solito non può essere abbastanza grande (non può contenere tutti i dati che si vogliono usare). Quando la cache si riempie, i dati che non sono stati usati per il tempo più lungo vengono scartati e la memoria liberata viene usata per quelli nuovi.

[3] Eccetto un disco di RAM, per ovvie ragioni.

La bufferizzazione del disco vale anche in scrittura; da una parte i dati scritti vengono spessi letti di nuovo (come per un codice sorgente che viene salvato in un file e poi letto dal compilatore), quindi mettere in cache i dati che vengono scritti è una buona idea; dall'altra, solo mettendo i dati nella cache e non scrivendoli nel disco il programma che li scrive gira più veloce. La scrittura si può fare in background, senza rallentare gli altri programmi.

La maggior parte dei sistemi operativi usano le cache di buffer (anche se si possono chiamare in altri modi), ma non tutte funzionano con i principi descritti sopra. Alcune sono *write-through*: i dati vengono scritti tutti insieme nel disco (vengono mantenuti anche nella cache, naturalmente), mentre la cache si chiama *write-back* se la scrittura viene fatta in un secondo tempo. Il write-back è più efficiente del write-through, ma anche più soggetto ad errori: se la macchina ha un crash, viene tolta la corrente al momento sbagliato o il floppy viene rimosso dal drive prima che i dati nella cache vengano scritti, i cambiamenti nella cache vengono persi; ciò può anche comportare che il filesystem (se ce n'è uno) non sia pienamente funzionante, perché forse i dati non scritti contenevano cambiamenti importanti alle informazioni di archiviazione.

Per questo non dovreste mai spegnere il computer senza usare una corretta procedura di shutdown (vedere il Capitolo 6), rimuovere un floppy dal drive senza smontarlo (se era stato montato) o dopo che qualsiasi programma che lo stia usando abbia segnalato di aver finito e che il led del floppy non si sia spento. Il comando **sync** fa il *flush* del buffer, cioè forza la scrittura di tutti i dati nel disco, e può essere usato quando si vuole essere certi che tutto venga scritto al sicuro. Nei sistemi UNIX tradizionali c'è un programma chiamato **update** in background che fa un **sync** ogni 30 secondi, in modo che non è in genere necessario usare **sync**. Linux ha un daemon aggiuntivo, **bdflush**, che fa un **sync** più imperfetto con frequenza maggiore per evitare il blocco improvviso dovuto al pesante input/output del disco causato a volte da sync.

Sotto Linux **bdflush** viene avviato da **update**. Di solito non c'è ragione per preoccuparsene, ma se per qualche ragione **bdflush** muore il kernel vi avviserà e dovrete riavviarlo a mano (**/sbin/update**).

La cache in genere non fa il buffer dei file ma di blocchi, che sono le unità più piccole dell'input/output dei dischi (sotto Linux di solito sono di 1 kB). In questo modo vengono messe in cache anche le directory, i superblocchi, altri dati di archiviazione dei filesystem e dischi che contengono un filesystem.

L'efficacia di una cache è principalmente decisa dalla sua dimensione: una cache piccola è praticamente inutile, dato che conterrà talmente pochi dati che tutti i dati in cache vengono tolti prima di venir riutilizzati. La dimensione critica dipende da quanti dati vengono letti e scritti e da quanto spesso vengono riutilizzati gli stessi. L'unico modo di saperlo è sperimentare.

Se la cache è di dimensione fissa non è neanche bene che sia troppo grande, perché potrebbe rimpicciolire troppo la memoria libera e rendere necessario fare swap (che è ugualmente lento). Per usare in modo più efficiente possibile la memoria reale Linux usa automaticamente tutta la RAM libera per la cache di buffer, ma rimpicciolisce la cache automaticamente quando i programmi hanno bisogno di più memoria.

Sotto Linux non dovete fare niente per utilizzare la cache, che funziona completamente in automatico; dovete solo seguire le procedure corrette per fare shutdown e per rimuovere i floppy.

Capitolo 6

Avvio e spengimento del sistema

Start me up. Ah... you've got to... you've got to never, never never stop. Start it up. Ah... start it up, never, never, never. You make a grown man cry, you make a grown man cry

(Rolling Stones)

Questa sezione spiega cosa succede quando viene avviato e spento un sistema Linux e qual'è la maniera corretta di farlo. Se non si seguono le procedure esatte si possono corrompere o perdere dei file.

6.1 Introduzione all'avvio ed allo spengimento del sistema

L'atto di accendere un computer e caricare il suo sistema operativo[1] si chiama *boot*. Il nome proviene da un immagine del computer che si tira su dai nastri di partenza (bootstrap), ma l'atto in sé è leggermente più realistico.

[1] Sui primi computer non era sufficiente accendere, ma bisognava caricare a mano il sistema operativo. Questi attrezzi moderni fanno tutto da soli.

Durante il boot, il computer per prima cosa carica una piccola parte di codice, il *bootstrap loader*, di solito immagazzinato in una posizione fissa sull'hard disk o su un floppy, che a sua volta carica ed avvia il sistema operativo. La ragione per cui avviene questo processo a due fasi è che il sistema operativo è grosso e complicato, ma la prima parte di codice che viene caricata dal computer deve essere molto piccola (poche centinaia di byte) per evitare di avere codice residente (*firmware*) troppo complesso.

Computer diversi hanno processi di bootstrap diversi. Per i PC il computer (il suo BIOS) legge il primo settore (il *settore di boot*) del floppy o dell'hard disk, che contiene il bootstrap loader, e carica il sistema operativo da qualche altro punto del disco (o da qualche altra parte).

Dopo essere stato caricato, Linux inizializza l'hardware e i device driver e poi avvia **init**. **init** inizializza altri processi per permettere agli utenti di collegarsi e fare altre cose; i dettagli di questa parte saranno discussi più avanti.

Per spegnere un sistema Linux per prima cosa viene detto a tutti i processi di terminarsi (devono chiudere i file che avevano aperto e fare altre cose necessarie per mantenere pulito il filesystem), vengono smontati i filesystem e le aree di swap e infine viene stampato un messaggio sulla console che avvisa che si può togliere la corrente. Se non viene seguita la procedura, possono accadere cose terribili: la peggiore possibile è che la cache di buffer non venga svuotata, il che significa che tutti i dati in essa contenuti vengono persi e che il filesystem sul disco è inconsistente, e quindi forse inutilizzabile.

6.1.1 Il processo di boot più da vicino

Si può avviare Linux sia da un floppy che dall'hard disk. La sezione sull'installazione nella guida *Installation and Getting Started* [1] vi spiega come installare Linux in modo da poterlo avviare nel modo che volete.

Al boot di un PC, il BIOS fa vari test per controllare che tutto sia a posto[2] e poi inizia il vero boot: sceglie un disco (tipicamente il primo floppy, se ce n'è uno inserito, oppure il primo hard disk, se nel computer ce ne è installato uno, ma l'ordine può essere cambiato) e ne legge il primo settore, che si chiama *boot sector* ("settore principale di boot"), o per un hard disk anche *master*

[2] Questo processo si chiama *power on self test*, o POST, cioè "auto–test di avvio".

boot record, dato che un hard disk può contenere diverse partizioni, ciascuna con il proprio settore di boot.

Il settore di boot contiene un piccolo programma (abbastanza piccolo da entrare in un singolo settore) la cui responsabilità è di leggere il vero sistema operativo dall'hard disk ed avviarlo. Quando si avvia Linux da un floppy disk, il settore di boot contiene del codice che legge semplicemente le prime centinaia di blocchi (a seconda della reale grandezza del kernel, naturalmente) in un punto predeterminato della memoria. Su un floppy di boot di Linux non c'è un filesystem, ma il kernel è semplicemente immagazzinato in settori consecutivi, dato che questo semplifica il processo di caricamento. È possibile, comunque, fare il boot da floppy con un filesystem, usando LILO, il LInux LOader.

Quando si fa il boot dall'hard disk, il codice nel master boot record esamina la tabella delle partizioni (anch'essa nel master boot record), identifica la partizione attiva (quella che è contrassegnata come avviabile), legge il settore di boot di quella partizione e poi inizializza il codice che vi si trova. Il codice nel settore di boot di quella partizione fa la stessa cosa che il settore di boot di un dischetto: legge il kernel dalla partizione e lo inizializza. I dettagli cambiano, comunque, dato che di solito non è utile avere una partizione separata solo per l'immagine del kernel, quindi il codice nel settore di boot della partizione non può semplicemente leggere il disco in maniera sequenziale, ma deve trovare i settori dove li ha messi il filesystem. Ci sono diversi modi per aggirare il problema, ma il più comune è usare LILO (i dettagli sul suo utilizzo sono irrilevanti per questa discussione: vedere la documentazione di LILO per altre informazioni, è molto accurata).

Quando si fa il boot con LILO, normalmente il computer passa direttamente alla lettura e al caricamento del kernel di default. È anche possibile configurare LILO per poter scegliere il kernel da caricare da un elenco, o anche per caricare sistemi diversi da Linux, ed è possibile scegliere quale kernel o sistema operativo avviare al momento del boot. LILO può essere configurato in modo che, se si tiene premuto **alt**, **shift**, o **ctrl** al momento del boot (quando si carica LILO), chieda quale sistema avviare, con un tempo massimo configurabile dopo il quale avvia il kernel di default.

Con LILO, è anche possibile dare un *argomento da linea di comando* al kernel dopo il suo nome o quello del sistema operativo.

Sia il boot da floppy che quello da hard disk hanno i loro vantaggi, ma in genere fare il boot da hard disk è più comodo, dato che evita la noia di dover maneggiare i floppy; è anche più veloce, ma configurare il sistema in modo che faccia il boot da disco fisso può essere più problematico, quindi molti prima installano il sistema in modo che faccia il boot dal floppy e poi, quando il sistema è installato e funzionante, installano LILO e cominciano a fare il boot dall'hard disk.

Dopo che il kernel di Linux è stato caricato in memoria, in qualsiasi modo questo accada, ed è stato avviato, succede più o meno questo:

- Il kernel di Linux è installato compresso, quindi per prima cosa si decomprime. L'inizio dell'immagine del kernel contiene un programmino che fa proprio questo.

- Se avete una scheda super-VGA che viene riconosciuta da Linux e che ha delle modalità di testo speciali (ad esempio 100 colonne per 40 linee), Linux vi chiede quale modalità volete usare. In fase di compilazione del kernel è possibile preimpostare una modalità video, in modo che non vi venga posta questa domanda. La stessa cosa si può anche fare con LILO o **rdev**.

- Dopo ciò, il kernel controlla quale altro hardware è presente (hard disk, floppy, schede di rete, ecc.) e configura alcuni dei dispositivi; mentre lo fa, manda in output dei messaggi su quello che trova. Ad esempio, all'avvio del mio computer vedo una cosa del genere:

```
LILO boot:
Loading linux.
Console: colour EGA+ 80x25, 8 virtual consoles
Serial driver version 3. 94 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
lp_init: lp1 exists (0), using polling driver
Memory: 7332k/8192k available (300k kernel code, 384k reserved, 176k data)
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
Loopback device init
Warning WD8013 board not found at i/o = 280.
Math coprocessor using irq13 error reporting.
Partition check:
  hda: hda1 hda2 hda3
```

```
VFS: Mounted root (ext filesystem).
Linux version 0.99.pl9-1 (root@haven) 05/01/93 14:12:20
```

Il testo esatto è diverso su sistemi diversi e cambia a seconda dell'hardware, della versione di Linux che si sta usando e di come è configurata.

- A questo punto il kernel prova a montare il filesystem radice. Il punto in cui lo va a cercare è configurabile al momento della compilazione o in qualsiasi momento usando **rdev** o LILO. Il tipo di filesystem viene individuato automaticamente. Se per qualche motivo non si riesce a montare il filesystem radice, ad esempio perché non vi siete ricordati di inserire il driver corrispondente nel kernel, si ha un *kernel panic* e il sistema si blocca (non c'è molto che potrebbe fare, comunque).

  Il filesystem radice di solito viene montato in modalità a sola lettura (si può configurare nello stesso modo che per la partizione da montare): ciò rende possibile controllare il filesystem mentre lo si monta; non è una buona idea controllare un filesystem montato in modalità lettura-scrittura.

- Dopo ciò, il kernel inizializza il programma **init** (che si trova in `/sbin/init`) in background (diventerà sempre il processo numero 1). **init** ha vari compiti all'avvio; cosa fa di preciso dipende da come è configurato, vedere il Capitolo 7 per altre informazioni. Di sicuro inizializzerà alcuni demoni in background.

- **init** poi passa in modalità multi-utente, ed inizializza una **getty** per le console virtuali e le linee seriali. **getty** è il programma che permette agli utenti di collegarsi attraverso le console virtuali ed i terminali seriali. **init** può anche inizializzare altri programmi, a seconda di come è stato configurato.

- A questo punto il boot è completo ed il sistema sta funzionando normalmente.

6.2 Ancora sullo shutdown

È importante seguire le procedure corrette quando si spenge un sistema Linux. Se non lo fate il filesystem probabilmente si corromperà e i file diventeranno illeggibili. Questo perché Linux utilizza la cache del disco, che non scrive sul disco tutto insieme ma solo ad intervalli; un comportamento del genere migliora moltissimo la performance, ma sta anche a significare che se spengete semplicemente il computer d'improvviso la cache può contenere molti dati e quello che si trova sul disco può non essere un filesystem che funziona perfettamente (perché solo alcune cose vi sono state scritte).

Un'altra ragione per non spengere direttamente l'interruttore è che in un sistema multitasking ci possono essere molti processi attivi in background e interrompere l'alimentazione può essere piuttosto disastroso. Usando la corretta sequenza di shutdown vi assicurate che tutti i processi in background possano salvare i propri dati.

Il comando per spengere correttamente un sistema Linux è **shutdown**. Normalmente può essere usato in due modi:

- Se vi trovate su un sistema dove siete l'unico utente, il modo normale di usare shutdown è uscire da tutti i programmi, scollegarsi da tutte le console virtuali, collegarsi come root su una di esse (o restare collegati come root se già lo siete, ma dovreste passare nella directory radice, per evitare problemi nello smontare i filesystem) e dare il comando **shutdown -h now** (sostituite **now** con un segno + ed un numero di minuti se volete un ritardo, anche se di solito non è questo il caso in un sistema monoutente).

- In alternativa, se il vostro sistema ha molti utenti, usate il comando **shutdown -h +tempo messaggio**, dove `tempo` è il tempo in minuti prima che il sistema venga fermato, e `messaggio` è una breve spiegazione del perché il sistema viene spento.

  ```
  # shutdown -h +10 'Dobbiamo installare un nuovo
  disco.  Il sistema dovrebbe tornare in linea tra tre ore.'
  #
  ```

  In questo modo tutti verranno avvertiti che il sistema si fermerà tra dieci minuti, e che è meglio che si scolleghino o perderanno dei dati. L'avviso viene visualizzato su tutti i terminali su cui c'è un utente collegato, inclusi tutti gli xterm:

```
Broadcast message from root (ttyp0) Wed Aug  2 01:03:25 1995...

Dobbiamo installare un nuovo disco.  Il sistema dovrebbe
tornare in linea tra tre ore.
The system is going DOWN for system halt in 10 minutes !!
```

L'avviso viene ripetuto automaticamente alcune volte prima dello spegnimento, ad intervalli sempre più brevi.

Quando comincia il vero shutdown dopo i vari ritardi, vengono smontati tutti i filesystem (eccetto quello di radice), i processi utente (se qualcuno è ancora collegato) vengono uccisi, i daemon vengono fermati ed in generale tutto si mette a posto. Fatto questo, **init** stampa un messaggio che indica che potete spengere la macchina. Allora, e solo allora, le vostre dita si possono muovere nella direzione dell'interruttore.

Talvolta, anche se raramente sui buoni sistemi, è impossibile fare lo shutdown in maniera corretta. Ad esempio se si ha un kernel panic e il kernel muore, esplode, o in genere non si comporta come dovrebbe, può essere totalmente impossibile dare nuovi comandi, quindi è possibile che sia difficile fare bene lo shutdown e tutto quello che potete fare è sperare che non sia successo niente di grave e spengere il computer. Se i danni sono un po' meno pesanti (ad esempio qualcuno ha preso ad accettate la vostra tastiera) ed il kernel e il programma **update** stanno ancora girando normalmente, probabilmente è una buona idea aspettare un paio di minuti, dare la possibilità ad **update** di svuotare la cache e solo allora spengere il computer.

Alcuni per fare lo shutdown usano il comando **sync**[3] tre volte, aspettano che l'I/O del disco sia finito e spengono il computer. Se non ci sono programmi attivi è praticamente la stessa cosa che usare **shutdown**, ma non smonta i filesystem e ciò può provocare dei problemi con il codice dell'ext2fs di "filesystem pulito". Il metodo del **sync** triplo *non è raccomandato* (in caso ve lo stiate chiedendo, la ragione per tre **sync** è che nei primi UNIX, quando i comandi venivano digitati singolarmente, tre sync dava di solito tempo sufficiente a finire qualsiasi I/O del disco).

[3] **sync** fa il flush della cache di buffer.

6.3 Il reboot

Fare il reboot significa avviare di nuovo il sistema e si può ottenere facendo shutdown, togliendo la corrente e poi riattaccandola. Un metodo più semplice è chiedere a **shutdown** di farlo lui, invece di spengere soltanto. Lo si può fare usando l'opzione **-r** di **shutdown**, ad esempio dando il comando **shutdown -r now**.

La maggior parte dei sistemi Linux fanno **shutdown -r now** quando si preme **ctrl** – **alt** – **canc** sulla tastiera, in modo da riavviare il sistema. L'azione di **ctrl** – **alt** – **canc** è però configurabile e spesso è preferibile inserire un ritardo in sistemi multiutente. Per sistemi accessibili a chiunque si può configurare **ctrl** – **alt** – **canc** in modo che non abbia nessun effetto (vedi la Sezione 7.4 *Configurazioni speciali in /etc/inittab*).

6.4 Modalità utente singolo

Il comando **shutdown** può anche essere usato per portare il sistema in modalità utente singolo, in modo che nessuno possa collegarsi, tranne root che può usare la console; questo è utile per i compiti di amministrazione di sistema che non possono essere fatti mentre il sistema funziona normalmente.

6.5 Dischetti di avvio di emergenza

Non è sempre possibile avviare un computer dall'hard disk: ad esempio, se vi sbagliate a configurare LILO potete rendere inaccessibile il vostro sistema. Per situazioni come questa avete bisogno di un modo alternativo di avviare il computer, che sia sempre possibile (almeno finché funziona l'hardware): per i normali PC si usa il boot dal dischetto.

La maggior parte delle distribuzioni di Linux permettono di creare un *dischetto di avvio di emergenza* durante l'installazione. È una buona idea farlo, ma alcuni di questi dischetti contengono solo il kernel ed assumono che usiate i programmi di installazione della distribuzione per risolvere qualsiasi problema abbiate. Talvolta questi programmi non sono sufficienti: ad esempio, potreste voler recuperare dei file da backup fatti con software non presente nei dischetti di installazione.

È quindi possibile che sia necessario creare un dischetto di root personalizzato. Il *Bootdisk HOWTO* di Graham Chapman [7] contiene le istruzioni per farlo. Dovete naturalmente ricordarvi di mantenere aggiornati i vostri dischetti di boot e root di emergenza.

Non potete usare il lettore dei floppy su cui avete montato il floppy di root per altre cose; questo può essere un inconveniente se avete un solo lettore, ma se avete memoria sufficiente potete configurare il floppy di boot in modo che carichi il disco di root in un ramdisk (il kernel del floppy di boot deve essere configurato appositamente per questo). Una volta caricato in ramdisk il floppy di root, il drive dei floppy è libero e può essere usato per montare altri dischetti.

Capitolo 7

init

Uuno on numero yksi

(Slogan da una serie di film finlandesi.)

Questo capitolo descrive **init**, il primo processo a livello utente inizializzato dal kernel. **init** ha molti compiti importanti, come inizializzare le **getty** (in modo che gli utenti si possano collegare), implementare i runlevel e prendersi cura dei processi orfani; questo capitolo spiega come si configura e come usare i diversi runlevel.

7.1 init prima di tutto

init è uno di quei programmi essenziali per il funzionamento di un sistema Linux, ma che si possono quasi sempre ignorare. Una buona distribuzione di Linux avrà una configurazione di **init** che funzioni con la maggior parte dei sistemi e su questi non ci sarà bisogno di fargli niente. Di solito preoccuparsi di **init** serve solo se agganciate terminali seriali, modem in ingresso (non in uscita) o se volete cambiare il runlevel di default.

Quando il kernel si è avviato (cioè quando è stato caricato in memoria, si è inizializzato ed ha inizializzato tutti i driver di device, le strutture dati e cose del genere) finisce la sua parte del processo di boot facendo partire un programma a livello utente: **init**, che è quindi sempre il primo processo (e il suo numero di processo è sempre 1).

Il kernel cerca **init** in alcuni posti in cui viene tenuto storicamente, ma il suo posto corretto (su un sistema Linux) è `/sbin/init`. Se il kernel non trova **init** prova ad avviare **/bin/sh** e se non riesce a fare neanche quello l'avvio del sistema fallisce.

Quando **init** parte completa il processo di boot portando a termine diversi compiti di amministrazione, come il controllo dei filesystem, la pulizia di **/tmp**, l'avvio di svariati servizi e di una **getty** per ogni terminale e console virtuale a cui si possano collegare gli utenti (vedere il Capitolo 8).

Dopo che il sistema è partito correttamente, ogni volta che un utente si scollega **init** fa ripartire una **getty** nel terminale corrispondente (in modo che l'utente successivo possa ricollegarsi). **init** adotta anche i processi orfani: quando un processo avvia un processo figlio e muore prima di esso, quest'ultimo diventa figlio di **init**; è importante per varie ragioni tecniche, ma è bene saperlo, perché facilita la comprensione degli elenchi e degli alberi dei processi[1]. Sono disponibili anche alcune varianti di **init**: la maggior parte dei sistemi Linux usano **sysvinit** (scritto da Miquel van Smoorenburg), che è basato sull'**init** del System V. Le versioni BSD di Unix hanno un **init** diverso. La differenza principale sono i runlevel: il BSD non ce l'ha (almeno tradizionalmente), ma non è una differenza essenziale. Noi vedremo solo il **sysvinit**.

7.2 Configurazione di init per inizializzare le getty: il file `/etc/inittab`

Quando si avvia, **init** legge il file di configurazione `/etc/inittab`, e durante il funzionamento del sistema lo leggerà di nuovo se gli si manda il segnale HUP[2]; questa caratteristica rende inutile riavviare il sistema per attivare delle modifiche alla configurazione di **init**.

Il file `/etc/inittab` è piuttosto complicato; cominceremo con il caso semplice della configurazione delle linee per le **getty**. Le linee del file `/etc/inittab` consistono di quattro campi delimitati da due punti:

[1] **init** non può morire: non si può uccidere nemmeno con SIGKILL.
[2] Usando il comando **kill -HUP 1** da root, ad esempio.

```
id:runlevel:azione:processo
```

I campi sono descritti qui sotto. Oltre a ciò, `/etc/inittab` può contenere linee vuote e linee che cominciano con un cancelletto ('`#`'): entrambe vengono ignorate.

id
: Identifica la linea del file. Per le linee di **getty**, specifica il terminale su cui gira (il carattere dopo `/dev/tty` nel nome del file di device). Per altre linee non vuol dire niente (tranne che per le restrizioni sulla lunghezza), ma non deve essere univoco.

runlevel
: Il runlevel per cui va considerata la linea. I runlevel vengono indicati da una sola cifra, senza delimitatori (sono descritti nella prossima sezione).

azione
: L'azione che deve corrispondere alla linea, ad esempio `respawn` per riavviare il comando nel campo successivo quando esce, o `once` per farlo una volta sola.

processo
: Il comando da attivare.

Per aprire una **getty** sul primo terminale virtuale (`/dev/tty1`), in tutti i normali runlevel multiutente (2-5) si dovrebbe scrivere la seguente linea:

```
1:2345:respawn:/sbin/getty 9600 tty1
```

Il primo campo dice che questa è la linea di `/dev/tty1`, il secondo che si applica ai runlevel 2, 3, 4 e 5, il terzo campo significa che il comando deve essere riavviato quando esce (in modo che ci si possa collegare, scollegare e collegare di nuovo), l'ultimo è il comando che avvia la **getty** sul primo terminale virtuale[3].

Se voleste aggiungere ad un sistema dei terminali o delle linee modem in ingresso, dovreste aggiungere altre linee ad `/etc/inittab`, una per ciascun terminale o linea. Per altri dettagli consultate le pagine man di **init**, **inittab**, e **getty**.

[3] Versioni diverse di **getty** funzionano in maniera diversa. Consultate la pagina man, ed assicuratevi che sia quella giusta.

Se un comando non parte e **init** è configurato per farlo ripartire (con `restart`), userà moltissime risorse di sistema: **init** lo avvia, fallisce, lo avvia, fallisce, e così via all'infinito. Per evitare una cosa del genere, **init** tiene traccia di quanto spesso avvia un comando e se la frequenza cresce aspetterà cinque minuti prima di avviarlo di nuovo.

7.3 I runlevel

Un *runlevel* è uno stato di **init** e dell'intero sistema che definisce quali servizi di sistema sono operativi. I runlevel sono identificati da numeri: vedere la Tabella 7.1. Non c'è un parere comune per come usare i runlevel definiti a livello utente (dal 2 al 5): alcuni amministratori di sistema li utilizzano per definire quali sottosistemi funzionano, cioè se gira X, se la rete è operativa e così via, altri hanno tutti i sottosistemi che girano sempre e li avviano e li fermano individualmente senza cambiare runlevel, perché i runlevel sono uno strumento troppo grezzo per il controllo del sistema. Potete decidere da voi, ma può essere più semplice seguire come fa la vostra distribuzione di Linux.

Tabella 7.1: Numeri dei runlevel

0	Ferma il sistema.
1	Modalità monoutente (per l'amministrazione straordinaria).
2-5	Operazione normale (definiti dall'utente).
6	Reboot.

I runlevel sono configurati da `/etc/inittab` da linee come questa:

```
12:2:wait:/etc/init.d/rc 2
```

Il primo campo è un livello arbitrario, il secondo significa che la linea si applica per il runlevel 2. Il terzo campo dice che **init** dovrebbe avviare il programma che sta nel quarto campo una sola volta, quando si entra nel runlevel, e che **init** deve aspettare che sia completato. Il comando `/etc/init.d/rc` avvia qualsiasi comando sia necessario per far partire o interrompere i servizi per entrare nel runlevel 2.

Il comando nel quarto campo fa tutto il lavoro duro; inizializza i servizi che non stanno girando e ferma quelli che non dovrebbero farlo più nel nuovo runlevel. Quale è esattamente il comando e come sono configurati i runlevel dipende dalla distribuzione di Linux.

Quando parte, init cerca una linea in `/etc/inittab` che specifichi il runlevel di default:

```
id:2:initdefault:
```

Potete chiedere a **init** di entrare in un runlevel non di default all'avvio dando al kernel da linea di comando l'argomento `single` o `emergency`. Gli argomenti al kernel da linea di comando si danno ad esempio con LILO, il che permette di scegliere anche la modalità singolo utente (runlevel 1).

Mentre il sistema gira, il comando **telinit** può cambiare il runlevel; quando accade, **init** avvia il comando appropriato da `/etc/inittab`.

7.4 Configurazioni speciali in /etc/inittab

Il file `/etc/inittab` ha alcune caratteristiche speciali che permettono ad **init** di reagire a circostanze particolari; tali caratteristiche vengono segnate da speciali parole chiave nel terzo campo. Alcuni esempi:

`powerwait`
: Permette ad **init** di spegnere il sistema, quando viene a mancare la corrente. Assume l'uso di un UPS e di un software che lo controlli ed informi **init** che non c'è più corrente.

`ctrlaltdel`
: Permette ad **init** di riavviare il sistema quando l'utente preme **ctrl** – **alt** – **canc** sulla tastiera di console. Notare che l'amministratore di sistema può configurare la reazione a **ctrl** – **alt** – **canc** in modo che sia qualcos'altro, ad esempio che venga ignorata, se il sistema è accessibile al pubblico (o che avvii **nethack**.)

`sysinit`
: Il comando da far partire all'avvio del sistema; di solito fa cose come pulire la `/tmp`.

La lista qui sopra non è completa. Vedere la pagina man di `inittab` per controllare tutte le possibilità e per dettagli su come usare quelle riportate qui sopra.

7.5 Fare il boot in modalità utente singolo

Un runlevel importante è la *modalità a singolo utente* (runlevel 1), in cui solo l'amministratore di sistema usa la macchina e girano il minor numero possibile di servizi di sistema, compresi i login. La modalità a utente singolo è necessaria per portare avanti dei compiti di amministrazione[4], come usare **fsck** sulla partizione `/usr`, il che richiede che la partizione non sia montata, cosa che non può accadere a meno che quasi tutti i servizi di sistema non vengano uccisi.

Si può portare un sistema funzionante in modalità utente singolo usando il comando **telinit** per richiedere il runlevel 1. All'avvio ci si può entrare dando `single` o `emergency` sulla linea di comando del kernel: il kernel passa la linea di comando anche ad **init**, ed **init** riconosce così che non deve usare il runlevel di default (la linea di comando del kernel viene attivata in modi diversi a seconda di come avviate il sistema).

Fare il boot in modalità utente singolo talvolta è necessario per poter usare **fsck** a mano, prima che vengano montate delle partizioni o che venga toccata una partizione `/usr` corrotta (qualsiasi attività su un filesystem corrotto probabilmente lo rovinerà ancora di più, quindi bisogna usarci **fsck** il prima possibile).

Gli script di avvio che **init** usa attivano automaticamente la modalità utente singolo se l'**fsck** automatico al boot fallisce, per evitare che il sistema usi un filesystem talmente rovinato da non poter essere riparato da **fsck**. Una cosa simile è relativamente rara e di solito comporta che l'hard disk sia rotto o che si stia usando una versione del kernel sperimentale, ma è bene essere preparati.

Per sicurezza, un sistema configurato correttamente chiederà la password di root prima di avviare una shell in modalità utente singolo, altrimenti sarebbe facile entrare come root dando una linea a LILO (non funzionerà se il file

[4]Non dovrebbe essere usata per giocare a **nethack**.

`/etc/passwd` è stato rovinato da problemi al filesystem, naturalmente, ed il quel caso sarà meglio avere un floppy di boot a portata di mano).

Capitolo 8

Login e logout

Non mi interessa appartenere ad un club che accetta gente come me come membri.

(Groucho Marx)

Questo capitolo descrive cosa succede quando un utente si collega o fa logout: vengono descritte in dettaglio le varie interazioni di processi in background, file di log, file di configurazione e così via.

8.1 Login via terminale

La Figura 8.1 mostra come avvengono i login via terminale. Per prima cosa, **init** si assicura che ci sia un programma di **getty** per la connessione da terminale (o da console): **getty** ascolta al terminale ed aspetta che l'utente sia pronto a fare il login (di solito comporta che l'utente digiti qualcosa). Quando nota un utente, **getty** dà in output un messaggio di saluto (che si trova in `/etc/issue`), chiede il nome dell'utente e infine fa partire il programma **login**. **login** prende come parametro il nome dell'utente e chiede la password; se corrispondono, **login** avvia la shell configurata per l'utente, altrimenti esce semplicemente e termina il processo (probabilmente dopo aver dato all'utente un'altra possibilità di inserire il nome e la password). **init** nota che il processo è stato terminato e avvia una nuova **getty** per quel terminale.

Notare che il solo processo nuovo è quello creato da **init** (usando la chiamata di sistema `fork`); **getty** e **login** sostituiscono solo il programma che gira nel processo (usando la chiamata di sistema `exec`).

Per le linee seriali è necessario un programma separato per notare l'utente, dato che può essere (ed era tradizionalmente) complicato notare quando un terminale diventa attivo. **getty** si adatta anche alla velocità e ad altre impostazioni della connessione, cosa molto importante per le connessioni in ingresso, quando questi parametri cambiano da chiamata a chiamata.

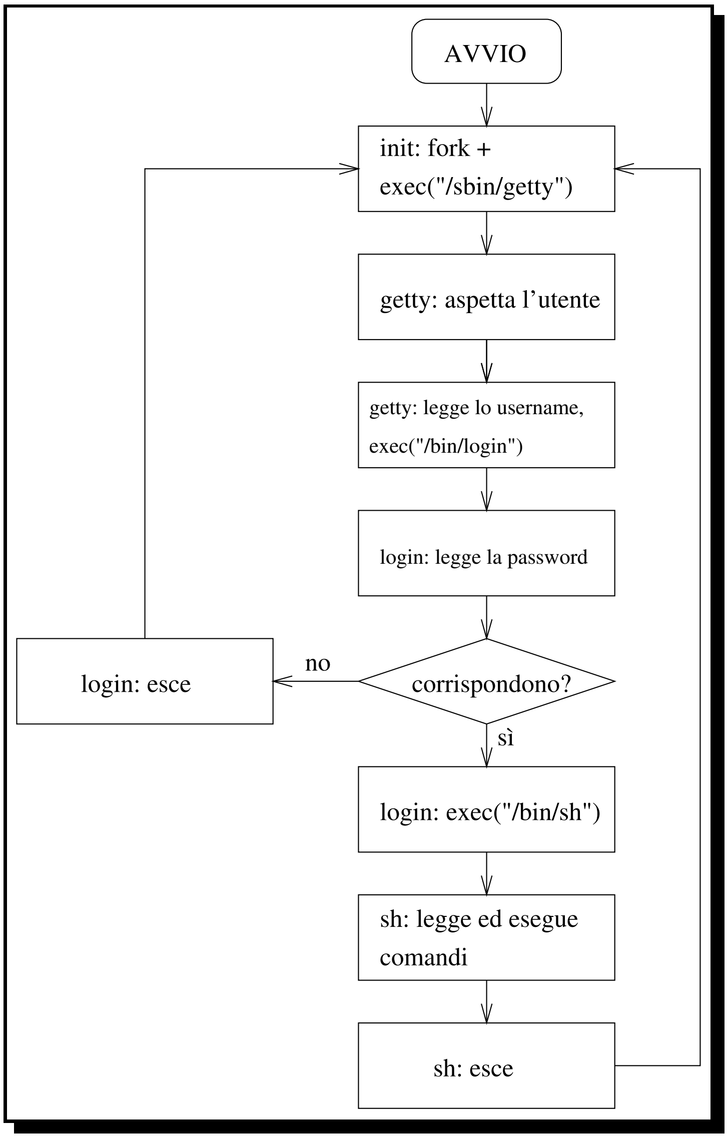
Esistono diverse versioni di **getty** ed **init**, tutte con i loro aspetti positivi e negativi. È una buona idea studiare tutte le versioni disponibili sul vostro sistema e anche le altre (potete usare la *Linux Software Map* [6] per cercarle). Se non avete accessi in entrata via linea seriale probabilmente non avete bisogno di preoccuparvi delle **getty**, ma **init** è comunque importante.

8.2 Login via rete

Due computer della stessa rete di solito sono collegati da un singolo cavo fisico. Quando comunicano in rete, i programmi in ciascun computer che prendono parte alla comunicazione sono collegati attraverso una *connessione virtuale*, una specie di cavo immaginario. Per quanto riguarda i programmi ai lati della connessione, hanno un monopolio del proprio cavo; comunque, dato che il cavo non è reale, solo immaginario, i sistemi operativi di entrambi i computer hanno diverse connessioni virtuali che condividono lo stesso cavo fisico. In questo modo, usando una sola connessione reale, diversi programmi possono comunicare tra di loro senza dover conoscere o doversi preoccupare delle altre comunicazioni. È anche possibile avere diversi computer che usano lo stesso cavo; le connessioni virtuali esistono tra due computer, e gli altri ignorano quelle di cui non fanno parte.

Questo è un modo complicato e troppo astratto per descrivere la situazione; può però essere abbastanza buono per capirle la ragione importante per cui i login via rete sono in qualche modo diversi da quelli normali: le connessioni virtuali vengono stabilite quando due programmi su computer diversi vogliono comunicare e, dato che in principio è possibile fare login su un qualsiasi com-

Figura 8.1 – *Login via terminale: l'interazione di **init**, **getty**, **login** e la shell.*



puter della rete da un qualsiasi altro, c'è un numero enorme di comunicazioni virtuali potenziali, quindi non è pratico inizializzare una **getty** per ogni login potenziale.

C'è un singolo processo di **inetd** (che corrisponde alle **getty**) che gestisce tutti i login di rete; quando nota un login di rete in arrivo (cioè nota che viene aperta una connessione virtuale da un altro computer), inizia un nuovo processo per gestirlo singolarmente. Il processo originale resta e continua ad aspettare altri login.

Per complicare un po' le cose, esiste più di un protocollo di comunicazione per i login di rete; i due più importanti sono **telnet** e **rlogin**. Oltre ai login, ci sono molte altre connessioni virtuali possibili (per FTP, Gopher, HTTP ed altri servizi di rete); sarebbe poco efficace avere un processo separato che aspetti un tipo particolare di connessione, quindi ce n'è uno solo che può riconoscere tutti i tipi e far partire il programma giusto per il servizio richiesto. Questo programma si chiama **inetd**; vedere la *Linux Network Administrators' Guide* [4] per altre informazioni.

8.3 Cosa fa login

login si occupa di autenticare l'utente (cioè di assicurarsi che il nome dell'utente e la password combacino), di configurare un ambiente iniziale impostando i permessi per la linea seriale e di inizializzare la shell.

Parte della configurazione iniziale consiste nel mandare in output il contenuto del file `/etc/motd` (per "*message of the day*", cioè "messaggio del giorno") e controllare la posta elettronica. Queste fasi possono essere disabilitate creando il file `.hushlogin` nella home directory dell'utente.

Se esiste il file `/etc/nologin` i login vengono disabilitati. Un file del genere viene di solito creato da **shutdown** e da comandi simili. **login** controlla la presenza di questo file e se esiste non accetterà i login, mandando in output il suo contenuto al terminale prima di uscire.

login tiene un log di tutti i tentativi di login falliti in un file di log di sistema (usando **syslog**). Tiene anche un log di tutti i login di root; entrambi possono essere utili per rintracciare gli intrusi.

Gli utenti collegati in ogni momento sono elencati in `/var/run/utmp`. Questo file è valido solo finché il sistema non viene riavviato o spento e viene ripulito quando il sistema si avvia; elenca ciascun utente ed il terminale (o la connessione di rete) che sta usando, insieme ad altre informazioni utili. I comandi **who**, **w** ed altri simili controllano in `utmp` per vedere chi è collegato.

Tutti i login che hanno avuto successo sono registrati in `/var/log/wtmp`: questo file crescerà senza limite, quindi deve essere ripulito regolarmente, ad esempio usando un job di **cron** settimanale[1]. Il comando **last** legge in `wtmp`.

Sia `utmp` che `wtmp` sono in formato binario (consultate la pagina man di `utmp`); sfortunatamente non conviene esaminarli senza usare programmi speciali.

8.4 La shell

Dopo l'autenticazione di **login**, all'utente viene presentato il prompt della shell: quale sia quella usata è scritto nel file `/etc/passwd`. La shell che viene avviata da **login** è sempre di tipo *interattivo di login*, in contrapposizione alle shell *non interattive*, che vengono avviate quando si utilizzano gli script di shell, ed a quelle *interattive non di login*.

Quando parte una shell di login interattiva, esegue automaticamente uno o più file predefiniti. Shell diverse eseguono file diversi; consultate la documentazione di ciascuna shell per altre informazioni.

La maggior parte delle shell eseguono prima un file globale e poi uno specifico per l'utente a cui appartiene la shell, ad esempio la Bourne shell (**/bin/sh**) e le sue derivate eseguono `/etc/profile` e poi `.profile` nella home directory dell'utente. `/etc/profile` permette all'amministratore di sistema di impostare un ambiente comune per gli utenti, specialmente per fare in modo che il PATH includa le directory di comandi locali oltre a quelle normali. D'altra parte, `.profile` permette agli utenti di personalizzare il proprio ambiente superando, se necessario, quello di default.

8.5 X e xdm

Se all'avvio del computer volete che non parta l'interfaccia testuale, ma si attivi direttamente quella grafica, potete configurare **xdm** in modo che sia

[1] Le buone distribuzioni di Linux lo fanno senza dover configurare niente.

possibile fare login in una finestra. Come per le sessioni testuali **login** autentica l'utente e fa partire una shell, nello stesso modo **xdm** fa l'autenticazione e fa partire una sessione, che comunemente è un window manager. Quando la sessione è terminata, **xdm** resetta il server X e (in maniera opzionale) riavvia il processo.

xdm permette anche di fare login non sulla macchina locale, ma su una remota. Per maggiori informazioni sull'argomento consultare la pagina man.

All'interno di una sessione di X, il corrispondente delle shell testuali si ritrova negli **xterm**. Un **xterm** avviato senza opzioni corrisponde ad una shell interattiva ma non di login: per la **bash** questo significa che viene letto ed eseguito il file `~/.bashrc`, valido per le shell interattive, ma non `/etc/profile` né `~/.bash_profile`, che si riferiscono alle sole shell di login.

Per avere all'interno di X una shell di login, in cui quindi siano attive tutte le impostazioni delle shell testuali, **xterm** va usato con l'opzione **-ls** (login shell).

8.6 Il controllo degli accessi

Tradizionalmente il database degli utenti è tenuto nel file `/etc/passwd`. Alcuni sistemi usano le *shadow password* ed hanno spostato le password in `/etc/shadow`. I siti con molti computer che condividono gli account usano il NIS o altri metodi per immagazzinare i database degli utenti: possono anche copiare automaticamente il database da una posizione centrale in tutti gli altri computer.

Il database degli utenti non contiene solo le password, ma anche altre informazioni sugli utenti, come il loro vero nome, le home directory e le shell di login. Queste informazioni devono essere pubbliche, in modo che chiunque possa leggerle, quindi le password sono criptate. Questo metodo ha il lato negativo che chiunque con accesso alle password criptate può usare un qualche metodo di decrittazione per leggerle, senza dover provare a collegarsi al computer; le shadow password tentano di evitare questa possibilità spostando le password in un altro file leggibile solo da root (sono comunque criptate).

Comunque, installare le shadow password su un sistema che non le supporta fin dall'inizio può essere difficoltoso.

Con o senza le shadow password, è importante assicurarsi che tutte le password del sistema siano valide, cioè non facilmente indovinabili. Per craccare le password si può usare il programma **crack**, e qualsiasi password che riesce a craccare per definizione non è valida; **crack** viene usato quindi, oltre che dagli intrusi, anche dai sistemisti che vogliono evitare password facili da indovinare. Si possono avere password valide anche usando il programma **passwd**; è più efficace in termini di cicli di CPU, dato che craccare le password è un lavoro che richiede molto calcolo.

Il database dei gruppi degli utenti viene tenuto in `/etc/group`; per i sistemi con le shadow password può anche esistere il file `/etc/shadow.group`.

root di solito non si può collegare dalla maggior parte dei terminali di rete, ma solo da quelli elencati nel file `/etc/securetty`; ciò rende necessario avere accesso fisico ad uno di questi terminali. È comunque possibile collegarsi ad un qualsiasi terminale come utente normale ed usare il comando **su** per diventare root.

Capitolo 9

La gestione degli account degli utenti

The similarities of sysadmins and drug dealers: both measure stuff in K's, and both have users.

(Vecchia barzelletta sull'informatica.)

Questo capitolo spiega come creare nuovi account utente, come modificare le proprietà di questi account, e come rimuoverli. Diversi sistemi Linux hanno strumenti diversi per farlo.

9.1 Che cos'è un account?

Quando un computer viene usato da molte persone è di solito necessario differenziare tra gli utenti, ad esempio in modo che i loro file privati possano essere mantenuti tali. È importante anche se il computer può essere usato da una sola persona alla volta, come con la maggior parte dei microcomputer[1]; quindi a ciascun utente viene dato un nome univoco e quello è il nome che viene usato per fare login.

[1]Potrebbe essere abbastanza imbarazzante se mia sorella potesse leggere le mie lettere d'amore.

Un utente non ha solo un nome, ma un *account*, che comprende tutti i file, le risorse e le informazioni che gli appartengono. Il termine ricorda le banche e in un sistema commerciale un account di solito corrisponde a del denaro, che svanisce a velocità diverse a seconda di quanto l'utente usa il sistema. Ad esempio lo spazio disco può avere un prezzo a megabyte o a giorno, ed il tempo CPU può avere un prezzo per secondo.

9.2 Come creare un utente

Il kernel di Linux tratta gli utenti come semplici numeri: ciascun utente viene identificato da un numero intero univoco, l'*identificativo utente* o *uid*, perché per un computer è più veloce e facile usare questi piuttosto che i nomi testuali. Un database separato fuori del kernel assegna un nome testuale, lo *username*, a ciascun identificativo utente, e contiene anche altre informazioni.

Per creare un utente bisogna aggiungere le informazioni ad esso relative al database e creargli una home directory. Può anche essere necessario educarlo ed impostargli un ambiente iniziale adatto.

La maggior parte delle distribuzioni di Linux hanno un programma per creare gli account; ce ne sono diversi disponibili. Due alternative da linea di comando sono **adduser** e **useradd**, ma ci possono essere anche degli strumenti grafici. Qualsiasi sia il programma, il risultato è che non c'è molto lavoro manuale da fare: anche se i dettagli sono molti ed intricati, questi programmi fanno sembrare tutto banale; comunque, la Sezione *Creazione manuale di un utente* descrive come farlo a mano.

9.2.1 `/etc/passwd` ed altri file informativi

Il database degli utenti principale in un sistema Unix è il file di testo `/etc/passwd` (chiamato *file delle password*), che elenca tutti i nomi utente validi e le informazioni ad essi associate. Il file ha una linea per ciascun nome utente e viene divisa in sette campi delimitati da due punti:

1. Nome utente.
2. Password, in forma criptata.
3. Id numerico dell'utente.

4. Id numerico del gruppo.
5. Nome completo o altra descrizione dell'account.
6. Home directory.
7. Shell di login (programma da avviare al login).

Il formato è spiegato in maggior dettaglio nella pagina di manuale di `passwd`.

Qualsiasi utente sul sistema può leggere il file delle password in modo da poter, ad esempio, trovare il nome di un altro utente: ciò significa che anche la password (il secondo campo) è disponibile per tutti. Il file delle password contiene le password in forma criptata, quindi in teoria non ci sono problemi; comunque la criptazione può essere decodificata, specialmente se le password sono deboli (brevi o che si trovano nel dizionario); non è quindi una buona idea tenere le password nel file delle password.

Molti sistemi Linux hanno le shadow password: un modo alternativo per tenere le password che vengono immagazzinate criptate in un file separato, `/etc/shadow`, leggibile solo da root. Il file `/etc/passwd` contiene solo un indicatore speciale nel secondo campo. Qualsiasi programma che debba verificare un utente è setuid e quindi può accedere al file delle shadow password. I programmi normali, che usano solo gli altri campi nel file delle password, non possono arrivare a leggerle[2].

9.2.2 Gli identificativi numerici per gli utenti e i gruppi

Sulla maggior parte dei sistemi non importa quali sono gli id numerici degli utenti e dei gruppi, ma se usate l'NFS (*Network File System*) dovete avere gli stessi uid e gid su tutti i sistemi; questo perché anche NFS identifica gli utenti con gli uid numerici. Se non usate l'NFS potete farli scegliere automaticamente al programma che crea gli account.

Se usate l'NFS dovrete inventare un meccanismo per sincronizzare le informazioni sugli account; un'alternativa è usare il sistema NIS (vedere [12]).

Comunque, dovreste cercare di non riutilizzare le uid numeriche (e gli username testuali), perché il nuovo padrone dell'uid (o dello username) potrebbe avere accesso ai file del vecchio utente (o alle mail, o che altro).

[2] Sì, vuol dire che il file delle password ha tutte le informazioni su un utente *tranne* le password: le meraviglie della tecnologia.

9.2.3 Ambiente iniziale: `/etc/skel`

Quando viene creata la home directory di un nuovo utente, viene inizializzata con i file dalla directory `/etc/skel`. L'amministratore di sistema può creare dei file in `/etc/skel` che daranno un ambiente di default per gli utenti; ad esempio può creare un file `/etc/skel/.profile` che imposta la variabile d'ambiente `EDITOR` ad un editor facile da usare.

Comunque di solito è meglio cercare di tenere `/etc/skel` il più piccolo possibile, perché sarebbe poi quasi impossibile aggiornare i file degli utenti esistenti. Ad esempio se cambia il nome dell'editor tutti gli utenti esistenti dovrebbero modificare il proprio `.profile`. L'amministratore di sistema può provare a farlo automaticamente con uno script, ma è quasi certo che danneggerebbe il file di qualcuno.

Quando possibile è meglio mettere le configurazioni globali nei file globali come `/etc/profile`; in questo modo è possibile aggiornarlo senza rovinare le impostazioni personali degli utenti.

9.2.4 Creazione manuale di un utente

Per creare a mano un account seguite questi passaggi:

- Modificate `/etc/passwd` con **vipw** e aggiungete una nuova linea per il nuovo account. Attenzione alla sintassi. *Non modificatelo direttamente con un editor!* **vipw** blocca il file, in modo che altri comandi non provino a modificarlo nello stesso tempo. Il campo della password dovrebbe essere impostato a '`*`', in modo che sia impossibile collegarsi.

- Nello stesso modo, modificate `/etc/group` con **vigr**, se dovete creare anche un nuovo gruppo.

- Create la home directory dell'utente con **mkdir**.

- Copiate i file da `/etc/skel` alla nuova home directory.

- Aggiustate l'owner e i permessi con **chown** e **chmod**. L'opzione **-R** è utilissima. I permessi corretti variano un po' da un sito all'altro, ma di solito questi comandi vanno bene:

```
cd /home/utente
chown -R utente.gruppo.
chmod -R go=u,go-w.
chmod go= .
```

- Impostate la password con **passwd**.

Dopo aver impostato la password nell'ultimo passaggio, l'account sarà funzionante. Non dovreste impostarla finché non avete fatto tutto il resto, altrimenti l'utente si potrebbe collegare inavvertitamente mentre state ancora copiando i file.

Qualche volta è necessario creare degli account fittizi[3] che non vengono usati da persone. Ad esempio, per creare un server FTP anonimo (in modo che chiunque possa scaricarne dei file senza dover avere un account) va creato un account con nome ftp. In tali casi di solito non c'è bisogno di impostare la password (l'ultimo passo qui sopra); invece, è meglio non farlo, in modo che nessuno possa usare l'account a meno di non diventare prima root, dato che root può diventare qualsiasi utente.

9.3 Modifica delle proprietà degli utenti

Ci sono alcuni comandi per modificare le varie proprietà di un account (cioè il campo rilevante in `/etc/passwd`):

chfn
: Cambia il campo del nome completo.

chsh
: Cambia la shell di login.

passwd
: Cambia la password.

Il superutente può usare questi comandi per cambiare le proprietà di qualsiasi account; gli utenti normali possono modificare solo quelle del proprio. Talvolta può essere necessario disabilitare questi comandi (con **chmod**) per gli utenti normali, ad esempio in un ambiente con molti utenti principianti.

[3] Utenti surreali?

Altri compiti vanno fatti a mano; ad esempio, per modificare il nome dell'utente va modificato `/etc/passwd` direttamente (ricordatevi, con **vipw**); nello stesso modo, per aggiungere o togliere l'utente da altri gruppi va modificato `/etc/group` (con **vigr**). Tali compiti sono però rari, e vanno fatti con cautela: ad esempio, se modificate il nome dell'utente la posta non arriverà più all'utente, a meno che non creiate un alias di posta[4].

9.4 Rimozione di un utente

Per rimuovere un utente prima vanno rimossi tutti i suoi file, i file e gli alias della posta, i job di stampa, di **cron** e di **at** e tutti gli altri riferimenti, poi si rimuovono le linee rilevanti dai file `/etc/passwd` e `/etc/group` (ricordatevi di rimuovere l'utente da tutti i gruppi di cui faceva parte). Può essere una buona idea disabilitare l'account (vedi sotto) prima di cominciare a rimuovere tutto, per evitare che l'utente lo usi mentre lo state facendo.

Ricordate che gli utenti possono avere dei file al di fuori della home directory: li potete trovare con il comando **find**:

```
find / -user utente
```

Notate comunque che il comando qui sopra prenderà *molto* tempo, se avete un disco grande. Se montate dei dischi via rete dovrete stare attenti a non bloccare la rete o il server.

Alcune distribuzioni di Linux hanno dei comandi speciali per farlo: cercate **deluser** o **userdel**; ma è facile anche farlo a mano e i comandi possono non fare tutto.

9.5 Disabilitazione temporanea di un utente

Talvolta è necessario disabilitare temporaneamente un account, senza rimuoverlo. Ad esempio, l'utente può non aver pagato l'abbonamento, oppure l'amministratore di sistema può sospettare che un cracker ne abbia la password.

[4] Il nome dell'utente può cambiare per un matrimonio, ad esempio, e l'utente potrebbe volere che il suo nome utente rifletta il suo nome nuovo.

Il modo migliore di disabilitare un account è cambiare la shell con un programma speciale che stampa solamente un messaggio. Così, chiunque provi a collegarsi nell'account non ci riuscirà e saprà il perché. Il messaggio può dire all'utente di contattare l'amministratore in modo da risolvere qualsiasi problema.

Sarebbe anche possibile cambiare il nome utente o la password in qualcos'altro, ma allora l'utente non saprebbe cosa sta succedendo. Utenti confusi significano maggior lavoro[5].

Un modo semplice di creare il programma speciale è scrivere degli 'script tail':

```
#!/usr/bin/tail +2
Questo account e' stato chiuso per questioni di sicurezza.
Chiamate il 555-1234 ed aspettate che arrivino gli uomini in nero.
```

I primi due caratteri ('`#!`') dicono al kernel che il resto della linea è il comando che deve essere usato per interpretare il file. Il comando **tail** in questo caso manda in output tutto tranne la prima linea nell'output standard.

Se l'utente `billg` è sospettato di un attacco alla sicurezza, l'amministratore di sistema farebbe una cosa del genere:

```
# chsh -s /usr/local/lib/no-login/security billg
# su - tester
Questo account e' stato chiuso per questioni di sicurezza.
Chiamate il 555-1234 ed aspettate che arrivino gli uomini in nero.
#
```

Lo scopo del comando **su** è di provare che il cambiamento ha funzionato, naturalmente.

Questi script devono essere tenuti in una directory separata, in modo che i loro nomi non interferiscano con i normali comandi utente.

[5] Ma può essere *veramente* divertente, se siete un BOFH.

Capitolo 10

I backup

L'hardware è indeterministicamente affidabile. Il software è deterministicamente inaffidabile. Le persone sono deterministicamente inaffidabili. La natura è deterministicamente affidabile.

Questo capitolo spiega perché, come e quando fare i backup e come recuperare dati da essi.

10.1 Sull'importanza delle copie di backup

I vostri dati sono preziosi. Vi costerebbe tempo e sforzo ricrearli e questo costa denaro o almeno nervosismo e lacrime; talvolta non si può neanche farlo, come ad esempio nel caso dei risultati di qualche esperimento. Dato che si tratta di un investimento, dovreste proteggerli e prendere provvedimenti per evitare di perderli.

Ci sono principalmente quattro ragioni per cui si possono perdere dei dati: guasti all'hardware, bachi del software, azione umana o disastri naturali[1]. Anche se l'hardware moderno tende ad essere abbastanza affidabile, si può comunque rompere senza una ragione apparente. Le parti hardware più critiche

[1] La quinta ragione è "qualcos'altro".

per i dati sono gli hard disk, che si affidano al fatto che deboli campi magnetici rimangano intatti in un mondo pieno di rumore elettromagnetico. Il software moderno non tende nemmeno ad essere affidabile; un programma solido come una roccia è un'eccezione, non la regola. Gli esseri umani sono piuttosto inaffidabili: possono fare uno sbaglio, o possono essere malvagi e distruggere dei dati volontariamente. La natura non sarà cattiva, ma può distruggere anche quando è buona. Tutto sommato, è un piccolo miracolo che qualcosa funzioni.

I backup sono un modo per proteggere l'investimento in dati: avendone diverse copie, non importa molto se una viene distrutta (il costo è solo quello di recuperarla dal backup).

È importante fare bene i backup; come tutte le cose correlate al mondo fisico, questi possono andare male prima o poi e parte del compito di farli è assicurarsi che funzionino: non è bello notare che il proprio backup non funziona nel momento del bisogno[2]. Aggiungendo la beffa al danno, il vostro computer può crashare proprio mentre state salvando i vostri dati: se avete un solo mezzo di backup, potrebbe distruggersi anche quello, lasciandovi solo con le ceneri fumanti del vostro duro lavoro[3], oppure potreste notare, nel momento in cui state recuperando i dati, che vi siete scordati di includere nel backup alcuni dati importanti, come il database degli utenti su un sito che ne ha 15.000. Meglio ancora, tutti i vostri backup potrebbero essere perfettamente funzionanti, ma l'ultimo lettore di nastri che poteva leggere quel tipo che usavate era quello che ora ha dentro un secchio d'acqua.

Quando si tratta di backup, la paranoia può essere un requisito richiesto.

10.2 La scelta del mezzo di backup

La decisione più importante sui backup è la scelta del mezzo: dovete considerarne il costo, l'affidabilità, la velocità, la disponibilità e l'utilizzabilità.

Il costo è importante, dato che dovreste preferibilmente avere molto più materiale di quello che vi serve per i dati. Un mezzo economico è di solito indispensabile.

L'affidabilità è estremamente importante, dato che un backup venuto male può fare piangere un uomo grande e grosso; un mezzo di backup deve essere in

[2] Non ridete, è successo a diverse persone.
[3] Mi è successo...

grado di mantenere i dati invariati per anni e il modo in cui lo si usa influisce sulla sua affidabilità: un hard disk è di solito molto affidabile, ma come mezzo di backup non lo è, se si trova nello stesso computer del disco di cui state facendo backup.

La velocità spesso non è molto importante, se si possono fare backup senza interazione: non importa che ci vogliano due ore se non c'è bisogno di alcuna supervisione. D'altra parte, se il backup non può essere fatto quando il computer non sarebbe comunque usato, allora anche la velocità è importante.

La disponibilità sul mercato è di certo necessaria, dato che non si può usare un mezzo di backup se non esiste. Meno ovvia è la necessità che il mezzo sia disponibile anche in futuro e su computer diversi dal vostro, altrimenti potreste non essere in grado di recuperare i dati dopo un disastro.

L'utilizzabilità è un fattore importante nella frequenza con cui vengono fatti i backup: più facile è, meglio è; un mezzo di backup non deve essere difficile o noioso da usare.

Le tipiche alternative sono i floppy e i nastri. I floppy sono molto economici, piuttosto affidabili, non molto veloci, si trovano facilmente, ma non possono essere usati per grandi quantità di dati. I nastri vanno da quelli economici a quelli piuttosto costosi, sono abbastanza affidabili, veloci, disponibili sul mercato e, a seconda della loro capacità, permettono di immagazzinare parecchi dati.

Ci sono altre alternative, di solito non molto comuni sul mercato, ma se questo non è un problema potenzialmente migliori sotto altri aspetti; ad esempio, i dischi magneto-ottici hanno dei vantaggi sia sui floppy (hanno alte capacità) che sui nastri (sono ad accesso casuale, rendendo così molto veloce il recupero di un singolo file).

Un altro mezzo utilizzato negli ultimi tempi è il CD-ROM, che ha una capacità media ed un prezzo piuttosto basso, ma ha la caratteristica di non poter essere riscritto, imponendo quindi una spesa aggiuntiva ad ogni backup. Il CD-ROM è quindi conveniente per backup non molto frequenti e per quantità di dati non molto alte.

10.3 La scelta dello strumento di backup

Esistono molti strumenti che possono essere usati per fare i backup: quelli usati tradizionalmente sotto UNIX sono **tar**, **cpio**, e **dump**, ma ne esistono molti altri (sia liberi che commerciali). La scelta del mezzo di backup può influenzare quella dello strumento da usare.

tar e **cpio** sono simili, e per lo più equivalenti dal punto di vista dei backup: entrambi possono immagazzinare dei file su nastro e recuperarli, entrambi riescono ad utilizzare quasi qualsiasi mezzo, dato che i driver del kernel si occupano della gestione a basso livello dei dispositivi e questi ultimi tendono a sembrare tutti uguali ai programmi a livello utente. Alcune versioni UNIX di **tar** e **cpio** possono avere problemi con file particolari (link simbolici, file di device, file con percorso molto lungo e così via), ma la versione Linux dovrebbe funzionare in tutti i casi.

dump è diverso nel fatto che legge direttamente i dati contenuti sul disco, senza utilizzare le strutture dati del filesystem. È anche scritto specificamente per i backup: **tar** e **cpio** in realtà sono fatti per archiviare file, anche se funzionano lo stesso per i backup.

Leggere il disco direttamente ha alcuni vantaggi: rende possibile fare copie di backup dei file senza modificare i timestamp; per **tar** e **cpio** bisognerebbe montare prima il filesystem in modalità a sola lettura; inoltre è più efficace se bisogna fare un backup completo, dato che lo si può fare senza spostare di molto la testina del disco. Lo svantaggio principale è che rende il programma di backup specifico per un tipo di filesystem: il programma **dump** per Linux riconosce solo il filesystem ext2.

dump supporta anche direttamente i livelli di backup (di cui parleremo più sotto): con **tar** e **cpio** bisogna implementarli con altri strumenti.

Un paragone tra gli altri strumenti di backup va oltre lo scopo di questo libro; la *Linux Software Map* elenca molti di quelli liberi.

10.4 Backup semplici

Un metodo di backup semplice è fare una copia di tutto una volta e poi copiare solo quei file che sono stati modificati dal precedente backup. Il primo viene chiamato *backup completo*, i successivi sono *backup incrementali*. Un backup completo è spesso più laborioso di quelli incrementali, dato che bisogna scrivere più dati sul nastro, e un solo nastro (o floppy) potrebbe non bastare, ma recuperare dati da backup incrementali può essere molto più faticoso che da un backup completo. Il recupero può essere ottimizzato facendo backup di tutto quello che è stato modificato dall'ultimo backup completo; in questo modo fare backup è un po' più laborioso, ma non ci dovrebbe mai essere bisogno di recuperare più che un backup completo e uno incrementale.

Se volete fare backup ogni giorno ed avete sei nastri, potete usare il nastro 1 per il primo backup completo (diciamo di venerdì) e i nastri da 2 a 5 per i backup incrementali (da lunedì a giovedì), poi fate un nuovo backup completo sul nastro 6 (il secondo venerdì) e ricominciate a fare backup incrementali con i nastri 2-5. Non dovete riscrivere il nastro 1 finché non avete un altro backup completo, nel caso vi succedesse qualcosa mentre state facendo il secondo; dopo che avrete usato il nastro 6 per il secondo backup completo, dovreste tenere il nastro 1 in qualche altro posto, in modo che se tutti gli altri nastri venissero distrutti da un incendio avreste ancora almeno un backup da usare. Quando dovrete fare il prossimo backup completo, prenderete il nastro 1 e lascerete il nastro 6 al suo posto.

Se avete più di sei nastri, potete usare quelli che avanzano per i backup completi. Ogni volta che fate un backup completo, usate il nastro più vecchio: in questo modo avrete backup di diverse settimane passate, il che va bene se volete trovare un file vecchio che avete cancellato o una vecchia versione di un altro.

10.4.1 Fare backup con tar

Si può fare facilmente un backup completo con **tar**:

```
# tar -create -file /dev/ftape /usr/src
tar: Removing leading / from absolute path names in the archive
#
```

L'esempio qui sopra usa la versione GNU di **tar** e le sue opzioni con nomi lunghi. La versione tradizionale di **tar** capisce soltanto le opzioni di un singolo carattere, la versione GNU può anche gestire backup che non entrano in un singolo nastro o floppy, ed anche percorsi molto lunghi: non tutte le versioni tradizionali lo fanno (Linux usa solo il **tar** GNU).

Se il vostro backup non entra in un nastro, dovete usare l'opzione **--multi-volume (-M)**:

```
# tar -cMf /dev/fd0H1440 /usr/src
tar: Removing leading / from absolute path names in the archive

Prepare volume #2 for /dev/fd0H1440 and hit return:
#
```

Notate che dovreste formattare i floppy prima di iniziare il backup, oppure usare un'altra finestra o terminale virtuale e farlo quando **tar** chiede un nuovo floppy.

Dopo fatto un backup, dovreste controllare che sia a posto, usando l'opzione **--compare (-d)**:

```
# tar --compare --verbose -f /dev/ftape
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
....
#
```

Se non controllate il backup noterete i vostri eventuali errori solo quando avrete perso i dati originali.

Potete fare un backup incrementale con **tar** usando l'opzione **--newer (-N)**:

```
# tar --create --newer '8 Sep 1995' -f /dev/ftape /usr/src -verbose
tar: Removing leading / from absolute path names in the archive
usr/src/
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/modules/
usr/src/linux-1.2.10-includes/include/asm-generic/
usr/src/linux-1.2.10-includes/include/asm-i386/
usr/src/linux-1.2.10-includes/include/asm-mips/
usr/src/linux-1.2.10-includes/include/asm-alpha/
usr/src/linux-1.2.10-includes/include/asm-m68k/
usr/src/linux-1.2.10-includes/include/asm-sparc/
usr/src/patch-1.2.11.gz
#
```

Sfortunatamente, **tar** non nota quando sono state modificate le informazioni dell'inode di un file, ad esempio i suoi bit di permesso, o il suo nome. Questo può essere aggirato usando **find** e paragonando lo stato corrente del filesystem con una lista di file compresa nel precedente backup. Sui siti ftp su Linux potete trovare degli script che fanno proprio questo.

10.4.2 Recuperare file con tar

L'opzione **--extract (-x)** di **tar** estrae dei file dall'archivio:

```
# tar --extract --same-permissions --verbose --file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
....
#
```

Si possono anche estrarre file o directory specifiche (compresi tutti i file e le sottodirectory in esse contenuti) dandoli sulla linea di comando:

```
# tar xpvf /dev/fd0H1440 usr/src/linux/include/linux/hdreg.h
usr/src/linux/include/linux/hdreg.h
#
```

Usate l'opzione **--list (-t)**, se volete solo vedere quali file ci sono in un volume di backup:

```
# tar --list --file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
....
#
```

Notate che **tar** legge sempre i volumi di backup sequenzialmente, quindi per volumi grandi è piuttosto lento. Non è possibile, comunque, usare delle tecniche di accesso casuale quando si usa un'unità a nastro o qualche altro mezzo ad accesso sequenziale.

tar non gestisce bene i file che sono stati cancellati. Se dovete recuperare un sistema da un backup completo e uno incrementale, ed avete cancellato dei file tra i due backup, questi esisteranno di nuovo dopo il recupero, il che può essere un problema serio se questi file hanno dati importanti che non dovrebbero essere più disponibili.

10.5 Backup multilivello

Il semplice metodo di backup descritto nella precedente sezione è spesso adeguato per l'uso personale o per siti piccoli; per un uso più pesante sono migliori i backup multilivello.

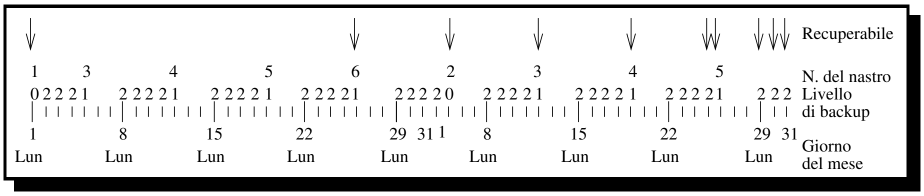
Il metodo semplice ha due livelli di backup: completo ed incrementale. Questo schema può essere generalizzato a qualsiasi numero di livelli: il backup completo sarebbe di livello 0, i diversi livelli di backup incrementale sarebbero i livelli 1, 2, 3 e così via. A ciascun livello incrementale si fa il backup di tutto quello che è cambiato dal precedente dello stesso livello o di un livello superiore.

Lo scopo di questo metodo è di permettere in modo poco dispendioso una storia di backup più lunga. Nell'esempio della Sezione precedente la storia di backup andava fino al precedente backup completo; si poteva estendere con più nastri, ma solo alla frequenza di una settimana per ogni nastro, il che poteva essere piuttosto dispendioso. Una storia di backup più lunga è utile, dato che i file cancellati o corrotti spesso non vengono notati per un lungo tempo. Anche una versione non molto aggiornata di un file spesso è meglio che niente.

Con il multilivello la storia di backup può venire estesa in maniera più economica. Ad esempio, se compriamo dieci nastri, potremmo usare il numero 1 e il 2 per i backup mensili (il primo venerdì del mese), i nastri dal 3 al 6 per i backup settimanali (gli altri venerdì: notate che ci possono essere cinque venerdì in un mese, quindi abbiamo bisogno di altri quattro nastri) e i nastri dal 7 al 10 per i backup giornalieri (dal lunedì al giovedì). Con solo altri quattro nastri possiamo estendere la storia di backup da due settimane (dopo che tutti i nastri giornalieri sono stati usati) a due mesi. È vero che non potremo recuperare tutte le versioni di ciascun file durante questi due mesi, ma quelle per cui possiamo farlo sono spesso più che sufficienti.

La Figura 10.1 mostra quali livelli di backup vengono usati in ciascun giorno e da quali backup si possono recuperare i file alla fine del mese. I livelli

Figura 10.1 – *Un esempio di backup multilivello.*



di backup possono essere usati anche per mantenere il tempo di recupero dei filesystem al minimo. Se avete molti backup incrementali con i numeri di livello che crescono costantemente dovrete usarli tutti se volete recuperare l'intero filesystem; potete invece usare dei livelli non monotoni e tenere basso il numero di backup da usare per il recupero dei file.

Per minimizzare il numero di nastri da cui recuperare potete usare un livello più basso per ciascun nastro incrementale. In questo caso, però, il tempo per fare backup cresce (ciascuna copia di backup contiene tutto quello che è stato modificato dal precedente backup completo). Uno schema migliore viene suggerito nella pagina man di **dump** e descritto in Tabella 10.1. Usate la seguente successione di livelli di backup: 3, 2, 5, 4, 7, 6, 9, 8, 9...In questo modo si mantengono bassi sia il tempo di backup che quello di recupero, ed al massimo dovrete recuperare due giorni di lavoro. Il numero di nastri per recuperare tutto dipende da quanto tempo fate passare tra i backup completi, ma è meno che negli schemi semplici. Uno schema strano può ridurre il lavoro necessario, ma implica più cose da seguire: dovete decidere se ne vale la pena.

dump ha incluso il supporto per i livelli di backup, per **tar** e **cpio** questo deve essere implementato con script di shell.

Tabella 10.1: Un efficace schema di backup usando molti livelli

Nastro	**Livello**	**Backup** (gg)	**Nastri di recupero**
1	0	n/a	1
2	3	1	1, 2
3	2	2	1, 3
4	5	1	1, 2, 4
5	4	2	1, 2, 5
6	7	1	1, 2, 5, 6
7	6	2	1, 2, 5, 7
8	9	1	1, 2, 5, 7, 8
9	8	2	1, 2, 5, 7, 9
10	9	1	1, 2, 5, 7, 9, 10
11	9	1	1, 2, 5, 7, 9, 10, 11
...	9	1	1, 2, 5, 7, 9, 10, 11, ...

10.6 Di cosa fare backup

Fate il backup di più cose possibili. L'eccezione principale è il software che si può facilmente reinstallare[4], che però può avere file di configurazione che è importante salvare, altrimenti dovrete riconfigurare tutto da capo. Un'altra eccezione importante è il filesystem `/proc`; dato che contiene solo dati che i kernel genera automaticamente, non è mai una buona idea farne il backup. Specialmente il file `/proc/kcore` è del tutto inutile, dato che è solo un'immagine della memoria fisica, ed è anche piuttosto grande.

A seconda dei casi potete fare o no il backup dello spool delle news, dei file di log e di altre cose in `/var`. Dovete decidere che cosa considerate importante.

La cosa ovvia di cui fare backup sono i file degli utenti (`/home`) e dei file di configurazione del sistema (`/etc`, ma probabilmente anche altre cose sparpagliate per il filesystem).

[4]Dovete decidere cosa vuol dire “facile”. Alcuni considerano facile reinstallare da dozzine di floppy.

10.7 Backup compressi

I backup prendono molto spazio, che può costare molti soldi; per ridurre lo spazio necessario si possono comprimere. Ci sono diversi modi di farlo: alcuni programmi hanno già un supporto interno per la compressione, ad esempio, l'opzione **--gzip (-z)** per il **tar** GNU manda in pipe l'intero backup attraverso il comando di compressione **gzip** prima di scriverlo sul mezzo di backup.

Sfortunatamente, i backup compressi possono fare danni. Per come funziona la compressione, se un singolo bit è sbagliato vengono persi tutti i dati compressi. Alcuni programmi di backup hanno la correzione degli errori automatica, ma non esiste un metodo che può gestire un gran numero di errori: ciò significa che se il backup è compresso come fa il **tar** GNU, con tutto l'output compresso in una singola unità, un solo errore fa perdere tutto il backup. I backup devono essere affidabili e questo metodo di compressione non è una buona idea.

Un metodo alternativo è comprimere ciascun file separatamente. Questo significa sempre che un file viene perso, ma tutti gli altri sono salvi. Il file perso sarebbe comunque corrotto, quindi questa situazione non è molto peggiore che non usare per niente la compressione. Il programma **afio** (una variante di **cpio**) supporta questo metodo.

La compressione può prendere del tempo, che può rallentare il programma di backup fino a renderlo troppo lento per scrivere su un nastro[5]. Questo si può evitare facendo il buffer dell'output (sia internamente, se il programma di backup è abbastanza intelligente, sia usando un altro programma), ma anche questo può non funzionare abbastanza bene; una cosa del genere dovrebbe però essere un problema solo su computer lenti.

[5] Se il drive dei nastri non riceve dati abbastanza velocemente, si deve fermare: questo rende il backup ancora più lento e non fa bene né al nastro né al drive.

Capitolo 11

Tenere il tempo

Time is an illusion. Lunchtime double so.

(Douglas Adams)

Questo capitolo spiega come un sistema Linux gestisce gli orari e descrive i passi necessari per evitare problemi. Di solito non c'è bisogno di fare niente a questo proposito, ma è bene capire come funziona.

11.1 I fusi orari

La misura del tempo è basata principalmente su fenomeni naturali, come l'alternanza di periodi di luce e buio causata dalla rotazione del pianeta. Il tempo totale impiegato da due periodi successivi è costante, ma la durata della luce e del buio varia. Una costante semplice è il mezzogiorno.

Il mezzogiorno è l'ora in cui il Sole è al suo punto più alto. Dato che la Terra è tonda[1], il mezzogiorno accade in momenti diversi a seconda del luogo. Questo porta al concetto di *ora locale*. Gli umani misurano il tempo in diverse unità, la maggior parte delle quali sono legate a fenomeni naturali come il mezzogiorno, ma finché rimanete nello stesso posto non vi importa che le ore locali siano diverse.

[1] Secondo moderne ricerche.

Nel momento in cui dovete comunicare con posti distanti vi accorgerete della necessità di avere un orario comune: nei tempi moderni, la maggior parte delle parti del mondo comunicano tra di loro, quindi è stato definito uno standard globale per l'orario che viene chiamato *ora universale* (UT o UTC, noto anche come *Greenwich Mean Time*, o GMT, dato che era l'ora locale di Greenwich, Inghilterra). Quando persone con orari diversi hanno bisogno di comunicare possono esprimere l'orario come ora universale, in modo che non ci sia confusione su quando le cose devono accadere.

Ogni orario locale si chiama fuso orario. Mentre la geografia permetterebbe a tutti i luoghi che hanno il mezzogiorno allo stesso momento di avere lo stesso fuso orario, la politica lo rende difficile, e per varie ragioni molti stati usano l'*ora legale*, cioè spostano gli orologi per avere più luce naturale mentre lavorano, e poi li rimettono a posto durante l'inverno, altri non lo fanno. Quelli che lo fanno non si trovano d'accordo sul giorno in cui spostare gli orologi e cambiano le regole di anno in anno, cosa che rende le conversioni decisamente non banali.

I fusi orari vengono chiamati nel modo migliore con il nome del luogo, o indicando la differenza tra l'ora locale e quella universale. Negli Stati Uniti ed in altre nazioni i fusi orari locali hanno un nome ed un'abbreviazione di tre lettere; le abbreviazioni non sono univoche, comunque, e non dovrebbero essere usate a meno che non venga nominato anche lo stato. È meglio parlare di ora locale diciamo di Helsinki, invece che di ora dell'Est Europeo, dato che non tutte le nazioni dell'Europa dell'Est usano le stesse regole.

Linux ha un pacchetto sui fusi orari che sa tutto ciò che bisogna sapere e che può essere facilmente aggiornato quando le regole cambiano. Gli amministratori di sistema devono solo selezionare il fuso giusto e ciascun utente può impostare il proprio (è importante, dato che molti lavorano su computer in nazioni diverse attraverso Internet). Quando cambiano le regole per l'ora legale nel vostro fuso orario, assicuratevi di aggiornare almeno quella parte del vostro sistema Linux. Oltre ad impostare il fuso orario del sistema e di aggiornare i file di dati dell'ora legale non ci si deve preoccupare molto degli orari.

11.2 Gli orologi hardware e software

I personal computer hanno un orologio hardware a batteria, che assicura che l'orologio funzioni anche se il resto del computer resta senza elettricità. L'orologio hardware può essere impostato dalla schermata di impostazione del BIOS o da qualsiasi sistema operativo stia girando.

Il kernel di Linux tiene traccia del tempo in maniera indipendente dall'orologio hardware. Durante il boot, Linux imposta il suo orologio allo stesso tempo di quello hardware; poi entrambi gli orologi girano indipendentemente. Linux mantiene un suo orologio perché guardare a quello hardware è lento e complicato.

L'orologio del kernel mostra sempre l'ora universale: in questo modo il kernel non ha bisogno di sapere niente dei fusi orari e la semplicità permette un'affidabilità più alta e rende più semplice aggiornare le informazioni sul fuso. Ciascun processo gestisce le conversioni sui fusi da solo (usando degli strumenti standard che fanno parte del pacchetto sui fusi orari).

L'orologio hardware può essere impostato sia sull'ora locale che su quella universale. Di solito è meglio averlo in ora universale, perché in questo modo non dovete modificare l'orologio hardware quando inizia o finisce l'ora legale (l'UTC non ha l'ora legale). Sfortunatamente qualche sistema operativo per PC, inclusi MS-DOS, Windows ed OS/2, assume che l'orologio hardware sia impostato sull'ora locale e quindi va modificato quando inizia e finisce il periodo di ora legale (altrimenti non mostrerebbe l'ora locale).

11.3 Impostazione e lettura dell'ora

Nel sistema Debian, il fuso orario di sistema va determinato dal link simbolico `/etc/localtime`, che è un collegamento al file di fuso orario che descriva il fuso orario locale. I file di fuso orario sono tenuti in `/usr/lib/zoneinfo`. Altre distribuzioni Linux possono fare diversamente.

Un utente deve modificare il suo fuso orario privato impostando la variabile d'ambiente `TZ`; se non è impostata viene considerato il fuso orario di sistema. La sintassi della variabile `TZ` viene descritta nella pagina man di `tzset`.

Il comando **date** mostra la data e l'ora corrente[2]. Ad esempio:

[2] Attenzione al comando **time**, che non mostra l'ora corrente.

```
$ date
Sun Jul 14 21:53:41 EET DST 1996
$
```

Questo corrisponde a Domenica, 14 Luglio 1996, circa le 10 meno 10 di sera nel fuso orario "EET DST" (che può stare per East European Time Daylight Savings Time). **date** può anche mostrare l'ora universale:

```
$ date -u
Sun Jul 14 18:53:42 UTC 1996
Sun Jul 14 18:53:42 UTC 1996
$
```

date si usa anche per impostare l'orologio software del kernel:

```
# date 07142157
Sun Jul 14 21:57:00 EET DST 1996
# date
Sun Jul 14 21:57:02 EET DST 1996
#
```

Vedere la pagina man di **date** per avere altri dettagli; la sintassi è un po' arcana. Solo root può impostare l'ora e, anche se ciascun utente può avere il proprio fuso orario, l'orologio è lo stesso per tutti.

date mostra o imposta soltanto l'orologio software; il comando **clock** sincronizza gli orologi hardware e software e viene usato all'avvio del sistema per leggere l'orologio hardware ed impostare quello software. Se dovete impostare entrambi gli orologi, prima impostate quello software usando **date** e poi quello hardware con **clock -w**.

L'opzione **-u** di **clock** dice che il clock hardware è in ora universale. *Dovete* usare correttamente l'opzione **-u**, altrimenti il vostro computer si confonderà.

L'orologio deve essere modificato con cautela, dato che molte parti di un sistema UNIX richiedono il suo funzionamento corretto; ad esempio, il daemon **cron** avvia dei comandi periodicamente e se modificate l'ora può confondersi e non sapere se deve avviare i comandi o no. Su un sistema UNIX di prima generazione, qualcuno ha impostato l'ora a venti anni nel futuro e **cron** voleva avviare tutti i comandi periodici dei venti anni tutti insieme; le versioni correnti di **cron** possono gestire eventualità del genere, ma dovreste comunque stare attenti: grandi salti in avanti o all'indietro sono più pericolosi di piccoli spostamenti.

11.4 Quando l'orologio si sbaglia

L'orologio software di Linux non è sempre accurato; viene tenuto aggiornato da un *timer interrupt* generato dall'hardware del PC. Se il sistema ha troppi processi che girano contemporaneamente può volerci troppo a generare l'interrupt e l'orologio software comincia ad andare indietro. L'orologio hardware gira indipendentemente e di solito è più accurato. Se fate spesso il boot del computer (come accade per la maggior parte dei sistemi che non fanno da server), normalmente questo terrà bene il tempo.

Se dovete rimettere l'orologio hardware, di solito è più facile riavviare, entrare nello schermo di impostazione del BIOS e farlo da lì; in questo modo si evitano tutti i problemi che possono insorgere modificando l'orologio di sistema. Se non potete farlo da BIOS, impostate l'orario nuovo con **date** e **clock** (in questo ordine), ma siate pronti a fare un reboot, se qualche parte del sistema comincia a fare strane cose.

Un computer in rete (anche se solo via modem) può controllare il suo orologio in automatico paragonandolo a quello di qualche altro computer. Se l'altro computer tiene il suo orologio all'ora precisa, allora entrambi possono farlo. Si può fare usando i comandi **rdate** e **netdate**; entrambi controllano l'ora di un computer remoto (**netdate** può gestirne più di uno) ed imposta il computer locale a quell'ora. Avviando regolarmente uno di questi comandi, il vostro computer terrà l'orario in modo accurato quanto quello del computer remoto.

Appendici

Appendice A

Glossario

> *Il Bibliotecario dell'Università Invisibile decise unilateralmente di aiutare la conoscenza producendo un Vocabolario Orang-utan/Umano. Ci stava lavorando da tre mesi. Non era facile: era arrivato a 'Oook'.*
>
> (Terry Pratchett, "Men At Arms")

Questa è una breve lista di definizioni di concetti relativi a Linux e all'amministrazione di sistema.

ambizione
: L'atto di scrivere frasi buffe nella speranza che vengano introdotte nel file di cookie di Linux.

chiamata di sistema
: I servizi forniti dal kernel ai programmi applicativi, ed il modo in cui vengono invocati. Vedere la sezione 2 delle pagine man.

daemon
: Un processo che aspetta in background, che di solito non viene notato a meno che qualcosa non lo faccia attivare. Ad esempio, il daemon **update** si sveglia ogni trenta secondi circa per svuotare la cache di buffer, e il daemon **sendmail** si sveglia ogni volta che qualcuno spedisce della posta.

filesystem
: I metodi e le strutture dati usate da un sistema operativo per tenere traccia dei file su un disco o su una partizione; il modo in cui i file sono organizzati sul

disco. Viene usato anche per indicare una partizione o un disco utilizzato per immagazzinare dei file o per il tipo del filesystem.

glossario
Un elenco di parole e di descrizioni. Da non confondere con il vocabolario, che è anch'esso un elenco di parole e descrizioni.

kernel
Parte di un sistema operativo che implementa l'interazione con l'hardware e la condivisione delle risorse. Vedere anche programma di sistema.

programma applicativo
Software che fa qualcosa di utile. Il risultato di usare un programma applicativo è il motivo per cui si compra il computer. Vedere anche programma di sistema, sistema operativo.

sistemi operativi
Software che condivide le risorse di un sistema informatico (processore, memoria, spazio disco, larghezza di banda di rete e così via) tra gli utenti e i programmi applicativi da essi usati. Controlla l'accesso al sistema per fornire sicurezza. Vedere anche kernel, programma di sistema, programma applicativo.

programma di sistema
Programmi che implementano funzionalità ad alto livello di un sistema operativo, cioè cose che non sono direttamente dipendenti dall'hardware. Possono talvolta richiedere privilegi speciali (ad esempio per consegnare la posta elettronica), ma spesso vengono considerati come parte del sistema operativo (ad esempio un compilatore). Vedere anche programma applicativo, kernel, sistema operativo.

sistema operativo
Software che condivide le risorse di un sistema informatico (processore, memoria, spazio disco, banda di rete e così via) tra gli utenti e i programmi applicativi da essi usati. Controlla l'accesso al sistema per fornire sicurezza.

Bibliografia

[1] Matt Welsh; *Installation and Getting Started Guide.*
Parte dell'LDP, disponibile elettronicamente presso `ftp://metalab.unc.edu /pub/Linux/docs/LDP/`; una guida omnicomprensiva su tutte le tematiche indispensabili per poter installare ed utilizzare un sistema Linux.

[2] Larry Greenfield; *Linux Users' Guide.*
Parte dell'LDP, disponibile elettronicamente presso `ftp://metalab.unc.edu /pub/Linux/docs/LDP/` anche in italiano; manuale di base per l'utilizzo di un sistema Linux, comprende anche le basi dell'uso dei principali editor.

[3] Lars Wirzenius; *Linux System Administrators' Guide.*
Parte dell'LDP, disponibile elettronicamente presso `ftp://metalab.unc.edu. /pub/Linux/docs/LDP/`.

[4] Olaf Kirch; *Linux Network Administrators' Guide.*
Parte dell'LDP, disponibile elettronicamente presso `ftp://metalab.unc.edu /pub/Linux/docs/LDP/`; manuale sull'uso delle reti con Linux, comprende anche le basi del networking.

[5] Michael K. Johnson; *Linux Kernel Hackers' Guide.*
Parte dell'LDP, disponibile elettronicamente presso `ftp://metalab.unc.edu /pub/Linux/docs/LDP/`; guida al kernel di Linux e punto di partenza per imparare le nozioni necessarie a diventare uno sviluppatore del kernel.

[6] Aaron Scrab; *Linux Software Map.*
È un elenco di tutto il software esistente sotto Linux. Si può ottenere presso `ftp://metalab.unc.edu/pub/Linux/docs/linux-software-map/`.

[7] Graham Chapman; *Bootdisk HOWTO.*
Disponibile assieme ad altri HOWTO di Linux presso `ftp://metalab.unc.edu/pub/Linux/docs/HOWTO/`; descrive come creare i propri dischetti di boot e root per Linux.

[8] Peter Anvin; *Linux Device List.*
Un elenco dei numeri dei dispositivi più o meno importanti per i dispositivi Linux. Ora è incluso nelle sorgenti del kernel.

[9] Stephen Tweedie, Alexei Vovenko; *Linux filesystem defragmenter.*
Disponibile elettronicamente presso `ftp://sunsite.unc.edu/pub/Linux/system/Filesystems/defrag-0.6.tar.gz`.

[10] Rémy Card; *The Second Extended Filesystem: Current State, Future Development.*
Trasparenze utilizzate durante l'intervento alla Seconda Conferenza Internazionale su Linux ed Internet tenutasi a Berlino nel Maggio del 1995.
Disponibile via FTP anonimo presso `ftp.ibp.fr`, nella directory `/pub/linux/packages/ext2fs/slides/berlin`.

[11] Paul Anderson; *Tips-HOWTO.*
Disponibile assieme ad altri HOWTO di Linux presso `ftp://metalab.unc.edu/pub/Linux/docs/HOWTO/`; elenca degli utili suggerimenti difficili da trovare nella documentazione.

[12] Thornsten Kukuk; *NIS-HOWTO.*
Disponibile assieme ad altri HOWTO di Linux presso `ftp://metalab.unc.edu/pub/Linux/docs/HOWTO/`; descrive la configurazione di Linux come client NIS(YP) o NIS+ e l'installazione di un server NIS.

[13] Evi Nemeth, Garth Snyder, Scott Seebass; *UNIX System Administration Handbook.*
Prentice-Hall (1989), ISBN 0-13-933441-6.
Da un Anonimo: non ho mai visto altri [manuali] con cui confrontarlo, così non so dire se lo raccomanderei particolarmente o meno. Copre, comunque, sia BSD che SYSV e per questo può risultare più utile ad un amministratore di sistema di Linux di un singolo libro che tratti esclusivamente o BSD o SYSV.

[14] Jerry Peek, Tim O'Reilly, Mike Loukide; *UNIX Power Tools.*
Bantam (1993), ISBN 0-679-79073-X.
Da un Anonimo: Non è una guida omnicomprensiva, ma include *molti* suggerimenti ed informazioni utili per un amministratore di sistema. Ha accluso inoltre un CD-ROM zeppo di programmi Unix utili.

[15] Daniel Quinlan; *Linux Filesystem Structure—Release 1.2.* (Marzo 1995)
Una descrizione ed una proposta per uno standard dell'albero delle directory di Linux, con l'intenzione di rendere più semplice la creazione di pacchetti di software e l'amministrazione di sistemi Linux facendo apparire i file in posizioni standard. Segue in maniera abbastanza pedissequa la consetudine Unix ed ha avuto il supporto della maggior parte delle distribuzioni Linux.
Disponibile via FTP presso `ftp.funet.fi`, nella directory `/pub/Linux/doc/fsstnd`.

[16] Eric Raymond; *The New Hacker's Dictionary.*
MIT Press (1991), ISBN 0-262-18145-2 (hc), 0-262-68069-6 (pbk). Un dizionario del gergo usato dagli *Hacker*. È una versione cartacea del *Jargon File*, il quale contiene tutto il testo del libro (in genere in una versione più aggiornata) e che è di pubblico dominio.