The Unix and Internet Fundamentals HOWTO

Eric S. Raymond v1.4, 25 settembre 1999

Questo documento descrive il funzionamento di base dei computer di classe PC, i sistemi operativi di tipo Unix e Internet senza far uso di un linguaggio troppo tecnico. Traduzione a cura di Mirko Nasato, *mnasato@iol.it* .

Indice

1 Introduzione

1.1 Scopo di questo documento

Questo documento vuole essere un aiuto per gli utenti di Linux e di Internet che stanno imparando dalla pratica. Anche se il “learning by doing” è un ottimo metodo per acquisire competenze specifiche, a volte lascia determinate lacune nella conoscenza delle basi che possono rendere difficile il pensiero creativo o la risoluzione efficace dei problemi, a causa della mancanza di un chiaro modello mentale relativo a cosa sta succedendo nella realtà.

Cercherò di descrivere con un linguaggio chiaro e semplice come funziona il tutto. La presentazione sarà calibrata per persone che usano Unix o Linux su hardware di classe PC. Di solito farò comunque riferimento semplicemente a ‘Unix’, dato che la maggior parte delle descrizioni vale anche per altre piattaforme e varianti Unix.

Assumerò che stiate usando un PC Intel. I dettagli differiscono leggermente se lavorate su un Alpha o un PowerPC o qualche altro computer Unix, ma i concetti di base sono gli stessi.

Non ripeterò le cose, quindi dovrete stare attenti, ma ciò significa anche che imparerete da ogni parola che leggete. È una buona idea limitarsi a dare una scorsa la prima volta che leggete; dovreste poi tornare indietro e rileggere alcune volte finché avrete digerito quello che avete imparato.

Questo è un documento in evoluzione. Intendo continuare ad aggiungere sezioni in risposta agli stimoli dei lettori, pertanto periodicamente dovreste tornare a rivederlo.

2 Novità

Novità nella versione 1.2: Sezione ‘Come fa il computer a immagazzinare le cose in memoria?’. Novità nella versione 1.3: Sezione ‘Che cosa accade con il log in?’ e ‘Proprietari dei file, autorizzazioni e sicurezza’.

2.1 Risorse correlate

Se state leggendo questo documento al fine di imparare come diventare un hacker, dovreste anche leggere la *How To Become A Hacker FAQ* `http://www.tuxedo.org/~esr/faqs/hacker-howto.html` . Contiene dei link ad altre risorse utili.

2.2 Nuove versioni di questo documento

Nuove versioni dello Unix and Internet Fundamentals HOWTO verranno periodicamente postate su *comp.os.linux.help* , *comp.os.linux.announce*

e *news.answers* `news:answers` . Saranno anche depositate su vari siti WWW e FTP dedicati a Linux, inclusa la LDP home page.

Potete vedere l'ultima versione sul World Wide Web all'URL `http://metalab.unc.edu/LDP/HOWTO/Fundamentals-HOWTO.html` .

2.3 Commenti, suggerimenti e correzioni

Se avete domande o commenti su questo documento sentitevi liberi di contattare Eric S. Raymond all'indirizzo *esr@thyrsus.com* . Qualsiasi suggerimento o critica sarà il benvenuto. Apprezzo particolarmente link a spiegazioni più dettagliate dei singoli concetti. Se trovate un errore per favore fatemelo sapere, in modo che lo possa correggere nella prossima versione. Grazie.

3 Anatomia di base del computer

Dentro al vostro computer c'è un chip processore che compie l'elaborazione vera e propria. C'è una memoria interna (quella che la gente DOS/Windows chiama "RAM" e la gente Unix spesso chiama "core"). Il processore e la memoria risiedono sulla *scheda madre*, che è il cuore del vostro computer.

Il vostro computer ha uno schermo e una tastiera. Ha dischi fissi e dischi floppy. Lo schermo e i dischi hanno *schede controller* che si attaccano sulla scheda madre e aiutano il computer a gestire questi dispositivi. (La tastiera è troppo semplice per aver bisogno di una scheda separata; il controller è costruito all'interno della tastiera stessa.)

Scenderemo più avanti in alcuni dei dettagli relativi al funzionamento di questi dispositivi. Per ora, ecco alcune cose di base da tenere a mente su come funzionano assieme:

Tutte le parti interne del vostro computer sono collegate da un *bus*. Fisicamente, il bus è quello dove si attaccano le schede controller (la scheda video, il controller del disco, una scheda audio se ce l'avete). Il bus è l'autostrada dei dati tra il processore, lo schermo, il disco e tutto il resto.

Il processore, che fa funzionare tutto il resto, in realtà non è in grado di vedere direttamente nessuno degli altri pezzi: deve comunicare con loro attraverso il bus. L'unico sottosistema al quale ha accesso veramente rapido, immediato, è la memoria (core). Perché i programmi siano eseguiti, dunque, devono essere *in memoria*.

Quando il vostro computer legge un programma o dei dati dal disco in effetti succede che il processore usa il bus per spedire una richiesta di lettura disco al controller del disco. Dopo un po' di tempo il controller del disco usa il bus per segnalare al computer che ha letto i dati e li ha messi in una certa locazione di memoria. Il processore può allora usare il bus per guardare in quella memoria.

Anche la tastiera e lo schermo comunicano con il processore attraverso il bus, ma in modi più semplici. Ne discuteremo più avanti. Per ora, ne sapete abbastanza per capire cosa succede quando accendete il vostro computer.

4 Cosa succede quando si accende un computer?

Un computer senza un programma in esecuzione è soltanto un ammasso inerte di componenti elettronici. La prima cosa che un computer deve fare quando viene acceso è far partire un programma speciale chiamato *sistema operativo*. Il compito del sistema operativo è aiutare gli altri programmi del computer a funzionare, gestendo gli intricati dettagli relativi al controllo dell'hardware del computer.

Il processo di avvio del sistema operativo si chiama *boot* (in origine era *bootstrap* e alludeva alla difficoltà di tirarsi su da solo, "by your bootstraps"). Il vostro computer sa come avviarsi perché le istruzioni per il boot sono incorporate in uno dei suoi chip, il BIOS (Basic Input/Output System).

Il chip BIOS gli dice di cercare uno speciale programma chiamato *boot loader* (quello di Linux si chiama LILO) che si trova in un posto predefinito del disco fisso con numero più basso (il *disco di avvio*). Il compito del boot loader è far partire il sistema operativo vero e proprio.

Per compiere quest'ultima operazione il loader cerca un *kernel*, lo carica in memoria e lo fa partire. Quando avviate Linux e vedete "LILO" sullo schermo, seguito da una riga di puntini, vuol dire che sta caricando il kernel. (Ogni puntino significa che ha caricato un altro *blocco del disco* di codice kernel).

(Vi potreste chiedere come mai il BIOS non carica il kernel direttamente: perché questo processo a due stadi con il boot loader? Beh, il BIOS non è molto intelligente. In effetti è proprio stupido, e Linux non lo usa più dopo la fase di avvio. Fu scritto in origine per i PC primitivi a 8 bit con dischi poco capienti e proprio non riesce ad accedere a una parte abbastanza grande del disco per caricare direttamente il kernel. La fase del boot loader consente anche di far partire diversi sistemi operativi da posti diversi del vostro disco, nella improbabile circostanza che Unix non vi soddisfi a sufficienza.)

Dopo essere partito, il kernel si guarda in giro, trova il resto dell'hardware e si prepara a far girare i programmi. Fa tutto questo guardando non nelle ordinarie locazioni di memoria ma piuttosto alle *porte I/O*, speciali indirizzi bus che probabilmente hanno schede controller di dispositivi che sono in ascolto in attesa di comandi. Il kernel non cerca a caso; ha molta conoscenza innata su cosa è probabile trovare dove, e su come i controller rispondono se sono presenti. Questo processo si chiama *autorilevamento*.

La maggior parte dei messaggi che vedete durante la fase di avvio sono del kernel che fa l'autorilevamento del vostro hardware attraverso le porte I/O, riconosce cosa ha a sua disposizione e si adatta al vostro computer. Il kernel di Linux è estremamente bravo in questo, meglio della maggior parte degli altri Unix e *molto* meglio del DOS o di Windows. Infatti, molti linuxiani della prima ora pensano che l'intelligenza del rilevamento all'avvio di Linux (che lo rende relativamente facile da installare) sia stata una delle principali ragioni che lo hanno fatto sfondare dal mucchio di esperimenti di Unix liberi, attraendo una massa critica di utenti.

Ma avere il kernel del tutto caricato e funzionante non è la fine del processo di boot; ne è solo il primo stadio (a volte chiamato *run level 1*, livello di esecuzione 1). A questo punto il kernel passa il controllo a un processo speciale chiamato 'init' che esegue diverse attività comuni.

Il primo compito del processo init è assicurarsi che i vostri dischi siano a posto. I file system dei dischi sono fragili: se vengono danneggiati da un malfunzionamento hardware o da un'improvvisa mancanza di elettricità, ci sono buoni motivi per compiere alcune operazioni di riaggiustamento prima che il vostro Unix sia tutto a posto. Parleremo più approfonditamente di questo più avanti, a proposito di 11.6 (come i file system si possono danneggiare).

Il passo successivo del kernel è far partire diversi *demoni*. Un demone (o daemon) è un programma quale uno spooler di stampa, un programma che attende di ricevere posta in arrivo oppure un server WWW che rimane latente in sottofondo, aspettando qualcosa da fare. Questi programmi speciali devono spesso coordinare diverse richieste che rischiano di entrare in conflitto. Sono demoni perché spesso è più facile scrivere un programma che gira costantemente e viene a conoscenza di tutte le richieste piuttosto che cercare di assicurarsi che un gruppo di copie (che girano tutte contemporaneamente, con ciascuna che processa una richiesta) non si ostacolino a vicenda. La particolare serie di demoni che il vostro sistema fa partire può variare, ma quasi certamente include uno spooler di stampa (un demone che fa da 'portinaio' per la vostra stampante).

Una volta che tutti i demoni sono avviati ci troviamo al *run level 2*. Il prossimo passo è prepararsi per gli utenti. Init avvia una copia di un programma chiamato `getty` per controllare la vostra console (e forse altre copie per controllare le porte seriali dial-in). Questo programma è quello che emette il prompt `login` alla vostra console. Siamo ora al *run level 3*, pronti per fare il log in e lanciare i programmi.

5 Che cosa accade con il log in?

Quando fate il log in (date un nome e la password) vi identificate a `getty` e al computer. Parte allora un altro programma chiamato (ovviamente) `login`, che controlla se siete autorizzati a usare quella macchina. Se non lo siete, il tentativo di log in viene rifiutato. Se lo siete, login compie qualche operazione di servizio e poi fa partire un interprete di comandi, la *shell*. (Sì, `getty` e `login` potrebbero essere un solo programma. Sono separati per motivi storici che qui non vale la pena approfondire.)

Ecco più in dettaglio che cosa accade prima che compaia la shell; sarà necessario comprenderlo più avanti quando parleremo di autorizzazioni dei file. Si viene identificati con un nome di login e password. Questo nome di login viene cercato in un file chiamato /etc/password, costituito da una sequenza di righe ciascuna delle quali descrive un account utente.

Uno di questi campi è una versione cifrata della password dell'account. Quello che inserite come password viene cifrato esattamente allo stesso modo e il programma `login` controlla se corrispondono. La sicurezza di questo metodo dipende dal fatto che, mentre è facile passare da una password in chiaro a una cifrata, l'inverso è molto difficile. Per cui, se qualcuno riesce a vedere la versione cifrata della vostra password non può comunque usare il vostro account. (Significa anche che se dimenticate la vostra password, non c'è modo di recuperarla, ma solamente di cambiarla in un'altra di vostra scelta.)

Una volta effettuato il log in con successo, otterrete tutti i privilegi associati al singolo account che state utilizzando. Potreste anche essere riconosciuti come appartenenti a un *group*. Un gruppo è un insieme di utenti impostato dall'amministratore e a cui è associato un nome. I gruppi possono avere privilegi indipendentemente dai privilegi dei loro membri. Un utente può appartenere a più gruppi. (Per maggiori dettagli sul funzionamento dei privilegi in Unix si veda la sezione su 11.5 ().)

(Si noti che, sebbene si fa normalmente riferimento agli utenti e ai gruppi per nome, essi sono in realtà memorizzati internamente come ID numerici. Il file password associa il vostro nome di account a un ID utente; il file `/etc/group` associa i nomi di gruppo agli ID numerici dei gruppi. I comandi che hanno a che fare con account e gruppi effettuano automaticamente la conversione.)

La vostra registrazione di account contiene anche la vostra *home directory*, il posto nel file system Unix che contiene i vostri file personali. Infine, la registrazione dell'account imposta anche la *shell*, l'interprete di comandi che `login` avvierà per accettare i vostri comandi.

6 Cosa succede quando si eseguono i programmi dalla shell?

La shell normale vi presenta il prompt '$' che vedete dopo il login (a meno che non lo abbiate personalizzato). Non parleremo della sintassi della shell e delle cose semplici che potete vedere da soli sullo schermo; daremo piuttosto uno sguardo dietro le quinte a quello che succede dal punto di vista del computer.

Dopo la fase di avvio, e prima che sia eseguito un programma, potete pensare al vostro computer come a un contenitore di un repertorio di processi che stanno tutti aspettando qualcosa da fare. Stanno tutti aspettando degli *eventi*. Un evento può essere voi che premete un tasto o muovete il mouse. Oppure, se il vostro computer è collegato a una rete, un evento può essere un pacchetto di dati che arriva lungo quella rete.

Il kernel è uno di questi processi. È uno speciale, perché controlla quando gli altri *processi utente* possono girare ed è normalmente l'unico processo con accesso diretto all'hardware del computer. Infatti, i processi utente devono fare richiesta al kernel quando vogliono ottenere un input dalla tastiera, scrivere sullo schermo, leggere o scrivere su disco o fare qualsiasi altra cosa che non sia macinare bit in memoria. Queste richieste sono note come *chiamate di sistema*.

Normalmente tutto l'I/O passa attraverso il kernel, così quest'ultimo può organizzare le operazioni e impedire che i processi si ostacolino a vicenda. Alcuni processi utente speciali hanno il permesso di aggirare il kernel, di solito per ottenere accesso diretto alle porte I/O. I server X (i programmi che gestiscono le richieste degli altri programmi di generare grafica sullo schermo, sulla maggior parte dei computer Unix) sono gli esempi più comuni al riguardo. Ma non siamo ancora arrivati a un server X; state guardando il prompt della shell su una console a caratteri.

La shell è solo un processo utente, e neppure uno tanto speciale. Attende che voi digitiate qualcosa, ascoltando (attraverso il kernel) sulle porte I/O della tastiera. Come il kernel vede che avete digitato qualcosa lo visualizza sullo schermo e poi lo passa alla shell. Quando il kernel vede un 'Invio' passa la vostra linea di testo alla shell. La shell tenta di interpretare questo testo come se si trattasse di comandi.

Diciamo che digitate 'ls' e Invio per invocare il programma Unix che elenca le directory. La shell applica le sue regole incorporate per indovinare che volete lanciare il comando eseguibile nel file '/bin/ls'. Fa una chiamata di sistema chiedendo al kernel di far partire /bin/ls come un nuovo processo *figlio* e di dargli accesso allo schermo e alla tastiera attraverso il kernel. Poi la shell va a dormire, aspettando che ls finisca.

Quando /bin/ls ha finito dice al kernel che ha terminato emettendo una chiamata di sistema *exit*. Il kernel allora sveglia la shell e le dice che può riprendere a girare. La shell emette un altro prompt e attende un'altra linea di input.

Tuttavia (supponiamo che stiate elencando una directory molto lunga) potrebbero succedere altre cose mentre 'ls' è in esecuzione. Potreste passare su un'altra console virtuale, fare il log in di là e iniziare una partita a Quake, per esempio. Oppure immaginate di essere collegati a Internet. Il vostro computer potrebbe spedire o ricevere posta mentre /bin/ls è in esecuzione.

7 Come funzionano i dispositivi di input e gli interrupt?

La tastiera è un dispositivo di input molto semplice: semplice perché genera piccole quantità di dati molto lentamente (per gli standard di un computer). Quando premete o rilasciate un tasto, il valore di questo evento viene segnalato attraverso il cavo della tastiera per far scattare un *interrupt hardware*.

È compito del sistema operativo stare attento a questi interrupt. Per ogni possibile tipo di interrupt c'è un *gestore dell'interrupt*, una parte del sistema operativo che immagazzina i dati a esso associati (come il valore del vostro premere/rilasciare il tasto) finché può essere processato.

Quello che effettivamente fa il gestore dell'interrupt della vostra tastiera è mettere il valore del tasto in un'area di sistema vicino al fondo della memoria. Là rimane a disposizione per ispezione quando il sistema operativo passa il controllo al programma che ritiene stia attualmente leggendo dalla tastiera.

Dispositivi di input più complessi come i dischi o le schede di rete funzionano in modo simile. Sopra abbiamo fatto il caso di un controller del disco che usa il bus per segnalare che una richiesta disco è stata ultimata. In realtà succede che il disco fa scattare un interrupt. Il gestore dell'interrupt del disco copia poi in memoria i dati ottenuti, a uso successivo da parte del programma che aveva fatto la richiesta.

A ogni tipo di interrupt è associato un *livello di priorità*. Gli interrupt con priorità più bassa (come gli eventi della tastiera) devono dare la precedenza agli interrupt con priorità più alta (come i tick dell'orologio o gli eventi del disco). Unix è progettato per dare alta priorità al tipo di eventi che hanno bisogno di essere processati rapidamente, in modo da mantenere fluida la risposta del computer.

Tra i messaggi d'avvio del vostro SO potete vedere dei riferimenti a numeri di *IRQ*. Forse sapete, senza capirne esattamente il perché, che uno dei modi più comuni di configurare male l'hardware è avere due dispositivi diversi che cercano di usare lo stesso IRQ.

Ecco la spiegazione. IRQ è l'abbreviazione di "Interrupt Request" (richiesta di interrupt). Il sistema operativo ha bisogno di sapere al momento dell'avvio quali interrupt numerati verranno usati da ciascun dispositivo hardware, in modo da poter associare a ciascuno il gestore appropriato. Se due dispositivi diversi cercano di usare lo stesso IRQ a volte gli interrupt verranno notificati al gestore sbagliato. Questo di solito provocherà quantomeno il blocco del dispositivo, ma può a volte confondere il SO a tal punto da farlo diventare instabile oppure mandarlo in crash.

8 Come fa il computer a fare diverse cose contemporaneamente?

Non lo fa, in realtà. I computer possono svolgere soltanto un task (o *processo*) alla volta. Ma un computer può cambiare task molto rapidamente e indurre i lenti esseri umani a pensare che sta facendo diverse cose contemporaneamente. Questo viene chiamato *timesharing* (condivisione di tempo).

Uno dei compiti del kernel è gestire il timesharing. Ha una parte chiamata *scheduler* (pianificatore) che contiene informazioni relative a tutti gli altri processi (a parte il kernel) del vostro repertorio. Ogni sessantesimo di secondo nel kernel fa scattare un timer e viene generato un clock di interrupt. Lo scheduler ferma qualunque processo sia attualmente in esecuzione, lo sospende sul posto e passa il controllo a un altro processo.

Un sessantesimo di secondo può non sembrare una grande quantità di tempo. Ma per i microprocessori odierni è sufficiente per eseguire decine di migliaia di istruzioni macchina, che si possono tradurre in una gran mole di lavoro. Quindi anche se ci sono molti processi ciascuno di essi può fare molte cose nella porzione di tempo a sua disposizione.

In pratica non sempre un programma ottiene la sua intera porzione di tempo. Se scatta un interrupt da un dispositivo I/O il kernel ferma effettivamente il task corrente, esegue il gestore dell'interrupt e poi ritorna al task corrente. Una tempesta di interrupt ad alta priorità può scombinare il normale funzionamento dei processi; questo fenomeno viene chiamato *thrashing* e per fortuna è molto difficile da indurre negli Unix moderni.

Infatti la velocità dei programmi solo molto di rado è limitata dalla quantità di tempo macchina a loro disposizione (ci sono alcune eccezioni a questa regola, quali il suono o la generazione di grafica 3D). Molto più spesso dei ritardi si generano quando il programma deve attendere dei dati da un disco o da una connessione di rete.

Un sistema operativo che può di norma gestire più processi simultaneamente è detto "multitasking". La famiglia di sistemi operativi Unix è stata progettata fin dall'inizio per il multitasking e lo fa molto bene, in modo molto più efficace rispetto a Windows o al Mac OS ai quali il multitasking è stato appiccicato a posteriori in seguito a un ripensamento e lo fanno in modo piuttosto povero. Il multitasking efficiente e affidabile costituisce buona parte di ciò che rende Linux superiore per le applicazioni di rete, le comunicazioni e i servizi Web.

9 Come fa il computer a evitare che i processi si intralcino tra loro?

Lo scheduler del kernel si prende cura di dividere il tempo tra i processi. Il vostro sistema operativo deve dividere tra i processi anche lo spazio, per evitare che non sconfinino oltre la porzione di memoria loro assegnata. Le operazioni compiute dal sistema operativo per risolvere questo problema si chiamano *gestione della memoria*.

Ogni processo del vostro repertorio ha la propria area di memoria core, come luogo dal quale eseguire il proprio codice e dove immagazzinare le variabili e i risultati. Potete pensare a questo insieme come formato da un *segmento codice*, di sola lettura (che contiene le istruzioni del processo), e da un *segmento dati* (che contiene tutte le variabili immagazzinate dal processo). Il segmento dati è sempre unico per ogni processo, mentre nel caso due processi usino lo stesso codice Unix automaticamente fa in modo che condividano un unico segmento codice, come misura di efficienza.

L'efficienza è importante, perché la memoria core è costosa. A volte non ne avete abbastanza per contenere per intero tutti i programmi che il computer sta eseguendo, specialmente se usate un grosso programma quale un server X. Per ovviare a questo problema Unix usa una strategia chiamata *memoria virtuale*. Non cerca di tenere in core tutti i dati e il codice di un processo. Tiene piuttosto caricato solo un *working set* relativamente piccolo; il resto dello stato del processo viene lasciato in uno speciale *spazio swap* sul vostro disco fisso.

Come i processi sono in esecuzione Unix tenta di anticipare i cambiamenti del working set per avere in memoria solo le parti che servono davvero. Riuscirci in modo efficace è ingegnoso e complesso, pertanto non cercherò di descriverlo tutto qui, ma si basa sul fatto che il codice e i riferimenti ai dati tendono a comparire a gruppi, ed è probabile che un nuovo gruppo si colleghi a luoghi vicini a quelli di uno precedente. Quindi se Unix tiene caricati i dati e il codice usati più di frequente (o di recente) di solito riuscirà a risparmiare del tempo.

Notate che in passato quel "A volte" di due paragrafi fa era un "Quasi sempre", perché la dimensione della memoria era tipicamente ridotta rispetto alla dimensione dei programmi in esecuzione, quindi il ricorso allo swap era frequente. Oggi la memoria è molto meno costosa e persino i computer di fascia bassa ne hanno parecchia. Sui moderni computer monoutente con 64MB di memoria e oltre è possibile eseguire X e un insieme tipico di programmi senza neppure ricorrere allo swap.

Anche in questa felice situazione la parte del sistema operativo chiamata *gestore della memoria* mantiene un importante ruolo da svolgere. Deve garantire che i programmi possano modificare soltanto il proprio segmento dati; deve cioè impedire che del codice difettoso o malizioso in un programma rovini i dati di altri programmi. A questo scopo tiene una tabella dei segmenti dati e codice. La tabella è aggiornata non appena un processo richiede più memoria oppure libera memoria (quest'ultimo caso si verifica di solito all'uscita dal programma).

Questa tabella è usata per passare comandi a una parte specializzata dell'hardware sottostante chiamata *MMU* o *unità di gestione della memoria*. I processori moderni hanno MMU incorporate. La MMU ha la peculiare capacità di porre dei delimitatori attorno alle aree di memoria, in modo che un riferimento che sconfina venga rifiutato e faccia scattare uno speciale interrupt.

Se avete mai visto un messaggio del tipo "Segmentation fault", "core dumped" o qualcosa del genere, questo è esattamente quello che è successo: un tentativo da parte del programma in esecuzione di accedere alla memoria al di fuori dal proprio segmento ha fatto scattare un interrupt fatale. Questo rivela un bug nel codice del programma; il *core dump* (scarico della memoria) che lascia dietro di sé costituisce una informazione diagnostica volta ad aiutare il programmatore nell'individuazione del problema.

C'è un altro aspetto che protegge i processi l'uno dall'altro, oltre alla limitazione della memoria a cui possono accedere. Si vuole anche poter controllare il loro accesso ai file in modo che un programma difettoso o malizioso non possa rovinare parti critiche del sistema. È per questo motivo che Unix possiede le 11.5 (autorizzazioni sui file) che vedremo in dettaglio più avanti.

10 Come fa il computer a immagazzinare le cose in memoria?

Probabilmente saprete che ogni cosa in un computer viene memorizzata come stringa di bit (binary digit; possiamo immaginarli come molti piccoli interruttori). Ora vedremo come questi bit vengano impiegati per rappresentare le lettere e i numeri che il computer manipola.

Prima di poter affrontare questo argomento, è necessario comprendere la *dimensione di parola* del computer. Si tratta della dimensione preferita dal computer per spostare unità di informazioni; tecnicamente è l'ampiezza dei *registri* dei processore, ovvero le aree che il processore utilizza per compiere calcoli logici e aritmetici. Quando leggiamo che i computer hanno dimensione in bit (per esempio "32-bit" o "64-bit") ecco che cosa si intende.

La maggior parte dei computer (compresi i PC 386, 486, Pentium e Pentium II) ha una dimensione di parola di 32 bit. Le vecchie macchine 286 lavoravano a 16. Mainframe vecchio stile spesso hanno parole di 36 bit. Pochi processori (come Alpha di quella che prima era la DEC e ora è Compaq) hanno parole di 64 bit. La parola di 64 bit diverrà più comune nei prossimi cinque anni; Intel sta progettando di sostiuire il Pentium II con un chip a 64 bit chiamato 'Merced'.

Il computer vede la memoria core come sequenza di parole numerate da zero in avanti, fino a valori molto grandi a seconda della dimensione della memoria. Tale valore è limitato dalla dimensione della parola, motivo per cui le vecchie macchine come i 286 dovevano svolgere complicati contorsionismi per indirizzare grandi quantità di memoria. Non li descriverò qui; procurano ancora degli incubi ai vecchi programmatori.

10.1 Numeri

I numeri sono rappresentati come parole o coppie di parole, a seconda della dimensione di parola del processore. Su macchine a 32 bit, la parola è la dimensione più comune.

L'aritmetica degli interi è simile ma non è esattamente identica alla matematica in base due. Il bit di ordine più basso è 1, il successivo 2, poi 4 e così via in notazione binaria. Ma i numeri dotati di segno sono rappresentati in notazione *complemento a due*. Il bit di ordine più alto è un *bit di segno* che rende negativa la quantità rappresentata, mentre ogni numero negativo può essere ottenuto dal valore positivo corrispondente invertendo tutti i bit. È per questo motivo che gli interi su una macchina a 32 bit devono essere compresi nell'intervallo tra -2^31 + 1 e 2^31 - 1 (dove ^ è l'operatore di elevamento a potenza, 2^3 = 8). Il 32-esimo bit è usato per il segno.

Alcuni linguaggi di programmazione danno accesso a una *aritmetica senza segno* ovvero una artimetica in base 2 con solo i numeri positivi e lo zero.

La maggior parte dei processori e alcuni linguaggi possono manipolare numeri in *virgola mobile* (funzionalità incorporata nel chip di tutti i processori recenti). I numeri in virgola mobile forniscono un intervallo più ampio degli interi e consentono di esprimere le frazioni. I modi in cui questo avviene sono diversi e un po' troppo complicati per essere affrontati in dettaglio in questa sede. Tuttavia, l'idea generale è molto simile alla cosiddetta 'notazione scientifica', dove si può scrivere (per esempio) 1.234 * 10^23; la codifica del numero viene divisa in una *mantissa* (1.234) e in un *esponente* (23) che indica le potenze di dieci.

10.2 Caratteri

I caratteri sono normalmente rappresentati come stringhe di sette bit, in una codifica chiamata ASCII (American Standard Code for Information Interchange). Sulle macchine moderne, ciascuno dei 128 caratteri ASCII è dato dai sette bit più bassi di un *ottetto* a 8 bit; gli ottetti sono riuniti in parole di memoria in modo che (per esempio) una stringa di sei caratteri occupi solamente due parole di memoria. Per vedere una mappa dei caratteri ASCII, scrivere 'man 7 ascii' al prompt di Unix.

Il paragrafo precedente, però, non è completamente corretto, per due ragioni. Quella minore è che il termine 'ottetto' è formalmente corretto ma raramente utilizzato; la maggior parte delle persone si riferisce a un ottetto come a un *byte* e ritiene che i byte siano lunghi otto bit. Per essere corretti, il termine 'byte' è più generale; per esempio, ci sono state macchine a 36 bit con byte di 9 bit (anche se probabilmente non capiterà più in futuro).

La ragione principale è, invece, che non tutto il mondo usa i codici ASCII. Infatti, molti paesi non possono usarli: mentre i codici ASCII funzionano bene per l'inglese americano, non contengono molte accentate e caratteri speciali necessari per le altre lingue. Persino l'inglese britannico ha il problema della mancanza di un segno per la sterlina.

Ci sono stati diversi tentativi di risolvere questo problema. Tutti fanno uso dell'ottavo bit non usato dai codici ASCII, che in questo modo risultano la metà inferiore di un set di 256 caratteri. Quello più largamente utilizzato è il set di caratteri 'Latin-1' (o più formalmente ISO 8859-1). Si tratta del set di caratteri predefinito per Linux, HTML e X. Microsoft Windows usa una versione mutante di Latin-1 che aggiunge alcuni caratteri come le virgolette destre e sinistre, in posizioni lasciate libere da Latin-1 per ragioni storiche (per una resoconto severo dei problemi che ha provocato, vedere la pagina *demoroniser* `http://www.fourmilab.ch/webtools/demoroniser/` .

Latin-1 gestisce le principali lingue europee, tra cui inglese, francese, tedesco, spagnolo, italiano, olandese, norvegese, svedese, danese. Tuttavia non è ancora sufficiente, per cui esistono altre serie di set di caratteri da Latin-2 a -9 per rappresentare il greco, l'arabo, l'ebraico e il serbo-croato. Per maggiori dettagli vedere la pagina *ISO alphabet soup* `http://www.utia.cas.cz/user_data/vs/documents/ISO-8859-X-charsets.html` .

La soluzione definitiva è uno standard enorme chiamato Unicode (e il suo gemello identico ISO/IEC 10646-1:1993). Unicode è identico a Latin-1 nella 256 posizioni più basse. Nello spazio successivo dei 16 bit comprende greco, cirillico, armeno, ebraico, arabo, devanagarico, bengalese, gurmukhi, gujarati, oriya, tamil, telugu, kannada, malese, tailandese, lao, georgiano, tibetano, giapponese kana, il set completo del coreano hangul moderno e un set unificato di ideogrammi cinesi/giapponesi/coreani (CJK). Per maggiori dettagli vedere la *Unicode Home Page* `http://www.unicode.org/` .

11 Come fa il computer a immagazzinare le cose su disco?

Quando leggete un disco fisso su Unix vedete un albero di nomi di file e directory. Normalmente non avrete bisogno di andare oltre, ma può essere utile avere maggiori dettagli se vi capita un crash del disco e dovete cercare di salvare dei file. Sfortunatamente non c'è un buon modo per descrivere l'organizzazione del disco dal livello dei file in giù, quindi dovrò partire dall'hardware e risalire.

11.1 Struttura di basso livello del disco e del file system

La superficie del vostro disco, dove vengono immagazzinati i dati, si divide in una sorta di bersaglio per il tiro a freccette: in tracce circolari che sono poi 'affettate' in settori. Dal momento che le tracce vicino al bordo esterno hanno area maggiore di quelle vicino al centro, le tracce esterne hanno più settori rispetto a quelle interne. Ogni settore (o *blocco del disco*) ha la stessa dimensione, che sui moderni Unix è generalmente pari a 1K binario (1024 parole da 8 bit). Ogni blocco è individuato da un indirizzo univoco, il *numero di blocco del disco*.

Unix divide il disco in *partizioni del disco*. Ogni partizione è formata da una serie continua di blocchi che vengono usati separatamente da quelli delle altre partizioni, come file system oppure come spazio swap. La partizione con numero più basso viene spesso trattata in modo speciale, come *partizione di avvio* dove si può mettere un kernel da far partire.

Ogni partizione è alternativamente uno *spazio swap*, usato per implementare 9 (memoria virtuale), oppure un *file system*, usato per contenere i file. Le partizioni swap sono trattate proprio come una sequenza lineare di blocchi. I file system, invece, hanno bisogno di un modo per associare i nomi dei file alle sequenze di blocchi disco. Dal momento che la dimensione dei file aumenta, diminuisce, si modifica nel tempo, i blocchi dati di un file non saranno una sequenza lineare ma potranno essere disseminati su tutta la sua partizione (dipende da dove il sistema operativo riesce a trovare un blocco libero quando gliene serve uno).

11.2 Nomi dei file e delle directory

All'interno di ciascun file system la corrispondenza tra i nomi e i blocchi viene assicurata da una struttura chiamata *i-node*. C'è un gruppo di questi elementi vicino al "fondo" (i blocchi a numerazione più bassa) di ciascun file system (quelli più bassi in assoluto sono usati a fini di manutenzione e di etichettatura, non ne parleremo qui). Ogni i-node individua un file. I blocchi dati dei file si trovano sotto gli i-node.

Ciascun i-node contiene una lista dei numeri di blocco disco relativi al file che individua. (Questa è una mezza verità, corretta solo per i file piccoli, ma il resto dei dettagli non è importante qui.) Notate che l'i-node *non* contiene il nome del file.

I nomi dei file si trovano nelle *strutture delle directory*. Una struttura della directory associa i nomi ai numeri i-node. Ecco perché, su Unix, un file può avere più nomi reali (o *hard link*); sono soltanto diverse voci di directory che puntano allo stesso i-node.

11.3 Mount point

Nel caso più semplice, tutto il vostro file system Unix si trova su di una sola partizione disco. Anche se questa situazione si ritrova in qualche piccolo sistema Unix personale, è inusuale. Più generalmente esso è suddiviso tra più partizioni disco, magari su diversi dischi fisici. Così, per esempio, il vostro sistema può avere una piccola partizione dove alloggia il kernel, una un po' più grande dove si trovano i programmi di utilità del SO e una molto più grande dove ci sono le directory personali degli utenti.

La sola partizione alla quale avrete accesso subito dopo l'avvio del sistema è la *partizione root*, che è (quasi sempre) quella dalla quale avete fatto il boot. Essa contiene la root directory del file system, il nodo superiore dal quale dipende tutto il resto.

Le altre partizioni del sistema devono collegarsi a questa root affinché tutto il vostro file system multipartizione sia accessibile. Circa a metà del processo di avvio, il vostro Unix renderà accessibili queste partizioni non root. Dovrà *montare* ciascuna di esse su una directory della partizione root.

Per esempio, se avete una directory chiamata '/usr', si tratta probabilmente di un mount point per una partizione che contiene molti programmi che fanno parte della distribuzione standard del vostro Unix ma che non sono necessari durante l'avvio iniziale.

11.4 Come viene cercato un file

Ora possiamo guardare al file system dall'alto al basso. Ecco cosa succede quando aprite un file (quale, ad esempio, `/home/esr/WWW/ldp/fundamentals.sgml`):

Il kernel parte dalla radice del vostro file system Unix (dalla partizione root). Cerca una directory chiamata 'home'. Di solito 'home' è un mount point per una grande partizione utente da qualche altra parte, così va di là. Nella struttura della directory di livello più alto di quella partizione utente cerca poi una voce chiamata 'esr' e ne estrae un numero di i-node. Va a quell'i-node, vede che si tratta di una struttura di directory e cerca 'WWW'. Estraendo *quell*'i-node, va alla corrispondente sottodirectory e cerca 'ldp'. Questo lo porta a un altro i-node di directory ancora. Aprendolo, trova il numero i-node di 'fundamentals.sgml'. Questo i-node non è una directory, ma contiene invece l'elenco dei blocchi disco associati al file.

11.5 Proprietari dei file, autorizzazioni e sicurezza

Per impedire ai programmi di intervenire accidentalmente o maliziosamente su dati su cui non dovrebbero, Unix usa le *autorizzazioni*. Queste vennero originariamente pensate per supportare il timesharing, proteggendo gli uni dagli altri utenti diversi sulla stessa macchina, quando ancora Unix veniva usato su costosi minicomputer condivisi.

Per comprendere le autorizzazioni sui file, occorre richiamare la descrizione di utenti e gruppi nella sezione 5 (Che cosa accade con il log in?). Ciascun file ha un utente proprietario e un gruppo proprietario. Inizialmente sono quelli del creatore del file; possono poi essere modificati con i programmi chown(1) e chgrp(1).

Le autorizzazioni fondamentali che possono essere associate a un file sono 'read' (autorizzazione a leggere i dati contenuti), 'write' (autorizzazione a modificarli) e 'execute' (autorizzazione a eseguirli come programma). Ciascun file ha tre set di autorizzazioni; uno per l'utente proprietario, uno per tutti gli utenti nel gruppo proprietario e uno per tutti gli altri. I 'privilegi' che si ottengono al momento del log in sono la possibilità di leggere, modificare ed eseguire quei file i cui bit di autorizzazione coincidono la propria ID utente o quella di un gruppo a cui si appartiene.

Per vedere come queste possono interagire e come le visualizza Unix, osserviamo alcuni elenchi di file su un sistema Unix ipotetico. Ecco un esempio:

```
snark:~$ ls -l notes
-rw-r--r--   1 esr      users         2993 Jun 17 11:00 notes
```

Si tratta di un file di dati ordinario. Il listato ci dice che il proprietario è l'utente 'esr', creato con il gruppo proprietario 'users'. Probabilmente la macchina su cui si trova mette per definizione tutti gli utenti ordinari in questo gruppo; altri gruppi che si vedranno comunemente su macchine con timesharing sono 'staff', 'admin', o 'wheel' (per ovvie ragioni, i gruppi non sono molto importanti su workstation a singolo utente o PC). Il vostro Unix potrebbe usare un gruppo di default differente, magari derivato dal vostro nome utente.

La stringa '-rw-r–r–' rappresenta i bit di autorizzazione per il file. Il primo trattino è la posizione del bit directory; se il file fosse stato una directory il bit sarebbe stato 'd'. Dopo di questo, le prime tre posizioni successive sono le autorizzazioni utente, le seconde tre le autorizzazioni del gruppo e le terze tre le autorizzazioni per gli altri (spesso chiamate autorizzazioni 'world'). Su questo file l'utente proprietario 'esr' può leggere e modificare il file, gli altri appartenenti al gruppo 'users' possono leggerlo e così tutti gli altri utenti. Si tratta di un set di autorizzazioni piuttosto tipiche per un file di dati ordinario.

Ora osserviamo un file con autorizzazioni molto diverse. Tale file è GCC, il compilatore C GNU.

```
snark:~$ ls -l /usr/bin/gcc
-rwxr-xr-x   3 root     bin          64796 Mar 21 16:41 /usr/bin/gcc
```

Questo file appartiene a un utente chiamato 'root' e ad un gruppo chiamato 'bin'; può essere modificato solo da root, ma letto ed eseguito da tutti. Si tratta di un proprietario e un set di autorizzazioni tipiche per un comando di sistema pre-installato. Il gruppo 'bin' esiste su alcuni Unix per raggruppare i comandi di sistema (il nome è una reliquia storica, abbreviazione di 'binary'). Il vostro Unix potrebbe usare invece un gruppo 'root' (non esattamente la stessa cosa dell'utente 'root'!).

L'utente 'root' è il nome convenzionale per l'ID utente con numero 0, un account speciale privilegiato che può scavalcare tutti i privilegi. L'accesso root è utile ma pericoloso; un errore di battitura quando si è collegati come root potrebbe rovinare file critici del sistema, cosa che non può avvenire con un account utente ordinario.

Poiché l'account root è così potente, il suo accesso dovrebbe essere sorvegliato attentamente. La password di root è il componente più critico nelle informazioni di sicurezza del sistema, e sarà quello che cercheranno di ottenere tutti i cracker e gli intrusi che verranno dopo di voi.

(Per quanto riguarda le password: non scrivetele su carta – e non scegliete password che possano essere indovinate facilmente, come il nome della/o vostra/o ragazza/o. È una pratica sorprendentemente comune che aiuta continuamente i cracker...)

Osserviamo ora un terzo caso:

```
snark:~$ ls -ld ~
drwxr-xr-x  89 esr      users         9216 Jun 27 11:29 /home2/esr
snark:~$
```

Questo file è una directory (osserviamo la 'd' in prima posizione). Vediamo che può essere modificata solo da esr, ma letta ed eseguita da tutti gli altri. Le autorizzazioni vengono interpretate in modo speciale sulle directory; esse controllano l'accesso ai file contenuti all'interno della directory.

Autorizzazione in lettura su una directory è semplice; significa semplicemente che potete esplorare la directory e aprire i file e le directory che contiene. L'autorizzazione in scrittura (modifica) da la possibilità di creare e cancellare file nella directory. Autorizzazione di esecuzione consente di effettuare *ricerche* nella directory – ovvero elencare il suo contenuto e vedere i nomi dei file e delle directory che contiene. A volte troverete directory che sono leggibili da tutti ma non eseguibili; questo significa che un utente qualunque può accedere a file e directory al suo interno, ma solamente se ne conosce il nome esatto.

Infine, osserviamo le autorizzazioni del programma login stesso.

```
snark:~$ ls -l /bin/login
-rwsr-xr-x   1 root     bin        20164 Apr 17 12:57 /bin/login
```

Possiede le autorizzazioni che ci aspetteremmo per un comando di sistema – tranne la 's' dove dovrebbe esserci il bit per l'autorizzazione in esecuzione del proprietario. Si tratta della manifestazione visibile di un tipo speciale di autorizzazione chiamata 'set-user-id' o *bit setuid*.

Il bit setuid è normalmente legato a programmi che hanno la necessità di dare agli utenti ordinari i privilegi di root, ma in modo controllato. Quando è impostato su un programma eseguibile, si acquistano i privilegi del proprietario di quel file finché si esegue quel programma, sia che essi coincidano con i nostri oppure no.

Come l'account root stesso, i programmi setuid sono utili ma pericolosi. Chiunque sia in grado di sovvertire o modificare un programma setuid che ha root come proprietario, può utilizzarlo per accedere alla shell con privilegi di root. Per questa ragione sulla maggior parte dei sistemi Unix, aprendo un file in scrittura automaticamente il suo bit setuid viene disattivato. Molti attacchi alla sicurezza su Unix tentano di scoprire bug nei programmi setuid, con lo scopo di sovvertirli. Amministratori di sistema attenti alla sicurezza sono quindi molto prudenti con questi programmi e riluttanti alla installazione di nuovi.

Ci sono un paio di importati dettagli che abbiamo sorvolato durante la discussione precedente sulle autorizzazioni; in particolare, come vengono assegnati l'utente e il gruppo proprietario quando viene creato un file per la prima volta. Il gruppo è importante poiché gli utenti possono essere membri di più gruppi, ma uno di essi (specificato nella voce dell'utente in `/etc/passwd`) è il *gruppo di default* dell'utente e normalmente possiederà i file creati dall'utente.

Per quanto riguarda i bit iniziali di autorizzazione, la faccenda è leggermente più complicata. Un programma che crea un file normalmente specificherà le autorizzazioni con cui dovrà partire. Queste, però, verranno modificate da una variabile nell'ambiente dell'utente chiamata *umask*. Umask speciica quali bit di autorizzazione *disattivare* quando crea un file; il valore più comune, e il default sulla maggior parte dei sistemi, è ——-w- o 002, che disattiva il bit di modifica per tutti gli utenti. Vedere la documentazione per il comando umask nella pagina di manuale della shell per maggiori dettagli.

11.6 Come le cose possono andare male

Prima accennavamo al fatto che i file system possono essere delicati. Ora sappiamo che per raggiungere un file dobbiamo fare il gioco della campana attraverso quella che può essere una catena arbitrariamente lunga di riferimenti i-node e directory. Supponiamo ora che sul vostro disco fisso si formi un punto danneggiato.

Se siete fortunati ciò vi farà perdere solo qualche file di dati. Se invece siete sfortunati, si potrebbe danneggiare una struttura di directory o un numero i-node e un intero sottoalbero del vostro sistema potrebbe rimanere pendente nel limbo. Oppure, peggio ancora, si potrebbe originare una struttura rovinata che punta in più modi allo stesso blocco disco o i-node. Un danneggiamento di questo tipo si può propagare a partire da una normale operazione sui file, facendo perdere tutti i dati collegati al punto danneggiato di origine.

Fortunatamente questo tipo di eventualità è divenuto abbastanza infrequente perché l'hardware dei dischi è più affidabile. Tuttavia, questo comporta che il vostro Unix voglia controllare periodicamente l'integrità del file system per assicurarsi che non ci sia nulla fuori posto. Gli Unix moderni compiono un rapido controllo dell'integrità di ciascuna partizione nella fase di avvio, giusto prima di montarle. Ogni tot riavvii fanno un controllo molto più approfondito che impiega qualche minuto in più.

Se tutto questo può far sembrare che Unix sia terribilmente complesso e incline a malfunzionamenti, può essere rassicurante sapere che questi controlli nella fase d'avvio tipicamente intercettano e correggono i problemi normali *prima* che diventino veramente disastrosi. Altri sistemi operativi non hanno questi strumenti, cosa che velocizza un po' l'avvio ma può mettervi molto di più nei pasticci quando cercate di fare un salvataggio a mano (e sempre assumendo che abbiate una copia delle Norton Utilities o simili, tanto per cominciare...).

12 Come funzionano i linguaggi per computer?

Abbiamo già visto 6 (come vengono eseguiti i programmi). Ogni programma in definitiva deve eseguire un flusso di byte che sono istruzioni nel *linguaggio macchina* del vostro computer. Ma gli esseri umani non se la cavano molto bene con il linguaggio macchina; riuscirci è divenuta un'arte rara, una magia nera persino tra gli hacker.

Quasi tutto il codice Unix, ad eccezione di una piccola porzione relativa all'interfaccia diretta con l'hardware nel kernel, viene oggi scritto in un *linguaggio di alto livello*. ('Alto livello' in questa espressione è un residuo storico volto a distinguerlo dai *linguaggi assembler* di 'basso livello', che sono fondamentalmente sottili involucri attorno al codice macchina.)

Ci sono diversi tipi di linguaggi di alto livello. Per affrontare l'argomento troverete utile tenere a mente che il *codice sorgente* di un programma (la versione creata dall'uomo, editabile) deve passare attraverso un qualche tipo di traduzione in codice macchina che il computer può effettivamente eseguire.

12.1 Linguaggi compilati

Il tipo più convenzionale di linguaggio è il *linguaggio compilato*. I linguaggi compilati vengono tradotti in file eseguibili di codice macchina binario da uno speciale programma chiamato (ovviamente) *compilatore*. Una volta che il codice binario è stato generato potete eseguirlo direttamente senza più guardare al codice sorgente. (La maggior parte del software è fornita come binari compilati a partire da codice che non vedete.)

I linguaggi compilati tendono a dare prestazioni eccellenti e hanno il più completo accesso al SO, ma tendono anche a essere difficili da programmare.

C, il linguaggio in cui Unix stesso è scritto, è di gran lunga il più importante tra questi (con la sua variante C++). FORTRAN è un altro linguaggio ancora usato tra gli ingegneri e gli scienziati ma di anni più vecchio e molto più primitivo. Nel mondo Unix nessun altro linguaggio compilato è nell'uso dominante. Al di fuori di esso, il COBOL è molto usato per il software finanziario e commerciale.

C'erano molti altri linguaggi compilati, ma la maggior parte di essi si sono estinti oppure sono strumenti strettamente di ricerca. Se siete nuovi sviluppatori Unix e usate un linguaggio compilato è estremamente probabile che questo sia il C o il C++.

12.2 Linguaggi interpretati

Un *linguaggio interpretato* dipende da un programma interprete che legge il codice sorgente e lo traduce al volo in calcoli e chiamate di sistema. Il sorgente deve essere reinterpretato (e l'interprete deve essere presente) ogni volta che il codice viene eseguito.

I linguaggi interpretati tendono a essere più lenti dei linguaggi compilati e spesso hanno accesso limitato al sistema operativo e all'hardware sottostanti. D'altra parte, essi tendono a essere più facili da programmare e più propensi a perdonare gli errori di codifica rispetto ai linguaggi compilati.

Molti programmi di utilità di Unix, inclusa la shell, bc(1), sed(1) e awk(1), sono in effetti piccoli linguaggi interpretati. I BASIC sono di solito interpretati. Così pure il Tcl. Storicamente, il più importante linguaggio interpretato è stato il LISP (un grande miglioramento rispetto ai suoi predecessori). Oggi il Perl è molto usato ed in costante crescita di popolarità.

12.3 Linguaggi a codice P

Dal 1990 è andato assumendo importanza crescente un tipo di linguaggi ibridi che usa sia la compilazione che l'interpretazione. I linguaggi a codice P sono come i linguaggi compilati nel senso che il sorgente viene tradotto in una forma binaria compatta che è ciò che viene realmente eseguito, ma che non è esattamente codice macchina. Si tratta invece di *pseudocodice* (o *codice P*) che è solitamente molto più semplice ma più potente di un vero linguaggio macchina. Quando eseguite il programma, interpretate il codice P.

Il codice P può girare velocemente quasi quanto un binario compilato (gli interpreti di codice P possono essere abbastanza semplici, leggeri e rapidi). Ma i linguaggi a codice P riescono a mantenere la flessibilità e la potenza di un buon interprete.

Importanti linguaggi a codice P includono Python e Java.

13 Come funziona Internet?

Per aiutarvi a capire come funziona Internet daremo un'occhiata alle cose che succedono quando fate una tipica operazione di Internet: indirizzate un browser alla prima pagina di questo documento, sul sito Web del Linux Documentation Project. L'indirizzo di questo documento è

```
http://metalab.unc.edu/LDP/HOWTO/Fundamentals.html
```

che significa che si trova nel file LDP/HOWTO/Fundamentals.html sotto la web directory dell'host metalab.unc.edu.

13.1 Nomi e locazioni

La prima cosa che il vostro browser deve fare è stabilire una connessione remota al computer dove si trova il documento. A tal fine deve prima trovare la locazione remota dell'*host* metalab.unc.edu ('host' è la forma breve di 'computer host' o 'host remoto'; metalab.unc.edu è un tipico *hostname*). La locazione corrispondente è in realtà un numero chiamato *indirizzo IP* (spiegheremo più avanti la parte 'IP' di questa espressione).

A questo scopo il vostro browser interroga un programma chiamato *name server*. Il name server può trovarsi sul vostro computer, ma è più probabile che giri su un computer del fornitore col quale il vostro computer dialoga. Quando vi collegate a un ISP una parte della procedura consiste quasi sicuramente nel dire al vostro software per Internet qual è l'indirizzo IP di un name server sulla rete dell'ISP.

I name server sui vari computer si parlano tra loro, scambiandosi e tenendo aggiornate tutte le informazioni necessarie per risolvere i nomi degli host (per metterli in corrispondenza con gli indirizzi IP). Il vostro name server può interrogare tre o quattro diversi siti sulla rete nel processo di risoluzione di metalab.unc.edu, ma di solito questo si verifica molto rapidamente (tipo in meno di un secondo).

Il name server dirà al vostro browser che l'indirizzo IP di Metalab è 152.2.22.81; a questo punto il vostro computer sarà in grado di scambiare direttamente bit con metalab.

13.2 Pacchetti e router

Quello che il browser vuole è mandare al server Web su Metalab un comando come questo:

```
GET /LDP/HOWTO/Fundamentals.html HTTP/1.0
```

Ecco cosa succede. Dal comando si costruisce un *pacchetto*, cioè un blocco di bit come un telegramma che è 'impacchettato' con tre cose importanti: l'*indirizzo di provenienza* (l'indirizzo IP del vostro computer), l'*indirizzo di destinazione* (152.2.22.81), e un *numero di servizio* o *numero di porta* (in questo caso 80) che indica che si tratta di una richiesta World Wide Web.

Il vostro computer spedisce allora il pacchetto lungo il cavo (la connessione modem al vostro ISP o rete locale) finché arriva a un computer specializzato chiamato *router*. Il router ha nella sua memoria una mappa di Internet, non sempre una completa, ma una che descrive completamente il vostro vicinato di rete e sa come raggiungere i router per altri circondari di Internet.

Il vostro pacchetto potrebbe passare attraverso svariati router lungo la strada per la sua destinazione. I router sono intelligenti. Guardano quanto tempo impiegano gli altri router per avvertire che hanno ricevuto un pacchetto. Usano questa informazione per dirigere il traffico verso i collegamenti veloci. La usano per accorgersi se un altro router (o un cavo) sono fuori servizio o irraggiungibili e quindi, se possibile, ovviare al problema trovando un'altra strada.

C'è una leggenda metropolitana secondo la quale Internet è stata progettata per sopravvivere alla guerra nucleare. Questo non è vero, ma la struttura di Internet è estremamente adatta a ottenere prestazioni affidabili anche con l'hardware precario che caratterizza questo mondo incerto. Questo deriva direttamente dal fatto che la sua intelligenza è distribuita tra migliaia di router piuttosto che riunita in poche enormi centrali (come la rete telefonica). Questo significa che i malfunzionamenti tendono a essere ben localizzati e la rete può aggirarli.

Una volta che il vostro pacchetto è giunto al computer di destinazione quest'ultimo usa il numero di servizio per inviare il pacchetto al server Web. Il server Web può capire a chi rispondere guardando l'indirizzo IP di provenienza del pacchetto con il comando. Quando il server Web restituisce questo documento lo suddivide in un certo numero di pacchetti. La dimensione dei pacchetti varia a seconda del mezzo di trasmissione sulla rete e del tipo di servizio.

13.3 TCP e IP

Per capire come vengono gestite le trasmissioni a pacchetti multipli, dovete sapere che Internet in realtà usa due protocolli, uno sovrapposto all'altro.

Il livello più basso, l'*IP* (Internet Protocol), sa come recapitare singoli pacchetti da un indirizzo di provenienza a un indirizzo di destinazione (è per questo che si chiamano indirizzi IP). Tuttavia l'IP non è affidabile: se un pacchetto si perde o cade i computer di origine e di destinazione possono non venirne mai a conoscenza. Nel gergo delle reti, l'IP è un protocollo *senza connessione*; il mittente si limita a far partire un pacchetto per il destinatario e non si aspetta un avviso di ricevuta.

L'IP è veloce ed economico, comunque. A volte veloce, economico e inaffidabile va bene. Quando giocate in rete a Doom o Quake, ogni pallottola è rappresentata da un pacchetto IP. Se alcune vengono perse, pazienza.

Il livello superiore, *TCP* (Transmission Control Protocol), vi dà affidabilità. Questi due computer negoziano una connessione TCP (cosa che fanno usando l'IP); il ricevente sa che deve spedire al mittente un avviso di ricevuta dei pacchetti che legge. Se il mittente non vede un avviso di ricevuta per un pacchetto entro un certo periodo di tempo (timeout) allora rispedisce quel pacchetto. Inoltre, il mittente attribuisce a ogni pacchetto TCP un numero di sequenza, che il ricevente può usare per riassemblare i pacchetti nel caso che risultino in disordine. (Cosa che si verifica se un collegamento della rete viene attivato o cade durante una connessione.)

I pacchetti TCP/IP contengono anche un checksum per consentire l'individuazione di dati rovinati da collegamenti difettosi. Così, dal punto di vista di chiunque usi il TCP/IP e i name server, sembra affidabile passare flussi di byte in coppie hostname/numero di servizio. Chi scrive i protocolli di rete non deve quasi mai pensare agli aspetti di basso livello relativi alla pacchettizzazione, al riassemblaggio dei pacchetti, al controllo degli errori, al checksum e alla ritrasmissione.

13.4 HTTP, un protocollo applicativo

Torniamo ora al nostro esempio. I browser e i server Web dialogano usando un *protocollo applicativo* che si appoggia al TCP/IP, usandolo semplicemente come un modo per passare stringhe di byte avanti e indietro. Questo protocollo è chiamato *HTTP* (Hyper-Text Trasfer Protocol, protocollo per il trasferimento di ipertesti) e abbiamo già visto un suo comando: il GET mostrato sopra.

Quando il comando GET arriva al server Web metalab.unc.edu con numero di servizio 80 verrà notificato al *demone server* che è in attesa sulla porta 80. La maggior parte dei servizi Internet sono implementati da demoni server che si limitano ad ascoltare sulle porte, attendono ed eseguono i comandi in arrivo.

Se il disegno di Internet ha una regola generale, questa è che tutte le parti dovrebbero essere il più possibile semplici e accessibili per gli esseri umani. L'HTTP, e i suoi simili (come il Simple Mail Transfer Protocol, *SMTP*, che viene usato per trasferire la posta elettronica tra gli host) tende a usare comandi in semplice testo stampabile che terminano con un codice di carriage return/line feed.

Questo è marginalmente inefficiente: in qualche circostanza potreste ottenere una velocità maggiore usando un protocollo binario di stretta codifica. Ma l'esperienza ha dimostrato che i vantaggi di avere comandi facili da descrivere e comprendere per gli esseri umani supera qualsiasi guadagno marginale di efficienza che si possa ottenere al prezzo di rendere le cose oscure e complicate.

Di conseguenza, quello che il demone server vi rispedisce via TCP/IP è anch'esso testo. L'inizio della risposta assomiglierà in qualche modo a questa (alcuni header sono stati omessi):

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 1998 18:43:35 GMT
Server: Apache/1.2.6 Red Hat
Last-Modified: Thu, 27 Aug 1998 17:55:15 GMT
Content-Length: 2982
Content-Type: text/html
```

Questi header saranno seguiti da una linea vuota e dal testo della pagina Web (dopodiché la connessione viene lasciata cadere). Il vostro browser si limita a visualizzare quella pagina. Gli header servono a spiegargli come (in particolare, l'header Content-Type gli dice che i dati restituiti sono veramente HTML).