Mario Sitta

Elementi di Informatica per Fisici

Facoltà di Scienze M. F. N.

Università del Piemonte Orientale – Alessandria

Mario Sitta
Elementi di Informatica per Fisici
Edizione 2.1, Ottobre 2003

L'Autore può essere contattato presso
Facoltà di Scienze M. F. N., C.so Borsalino 54, 15100 Alessandria, Italy

Questo testo è stato composto usando Plain TEX e con l'aiuto di alcune 'macro' sviluppate dall'Autore.

Introduzione

They who know, do. They who don't know, teach.

Anon.

I calcolatori elettronici hanno raggiunto uno sviluppo ed una diffusione davvero impressionanti. Anche nel campo della Fisica essi hanno innumerevoli applicazioni: studio e sviluppo di nuovi rivelatori, controllo e raccolta dei dati di apparati sperimentali, simulazioni numeriche di processi fisici, analisi statistica di dati sperimentali, e così via. Molte di queste applicazioni poi si ritrovano anche nel controllo dei processi produttivi industriali e nella ricerca applicata. E' quindi di importanza fondamentale che gli studenti del Corso di Laurea in Fisica apprendano fin dal primo anno per lo meno i fondamenti del funzionamento e dell'uso del calcolatore.

Lo scopo del modulo di Laboratorio di Informatica fisica I è appunto quello di fornire agli studenti i primi elementi di informatica. In particolare viene descritta l'organizzazione interna di un calcolatore elettronico, il funzionamento di un sistema operativo e la struttura di un linguaggio di programmazione. Come esempi pratici vengono utilizzati il sistema operativo Unix (nella versione Linux) ed il linguaggio C. Ogni argomento è trattato non in maniera puramente astratta, ma attraverso esempi e realizzazioni pratiche. Ovviamente il tempo disponibile non consente di esaminare nei singoli dettagli tutti gli argomenti, ma comunque sono forniti strumenti sufficienti affinché lo studente possa muoversi con autonomia nell'uso del calcolatore ed abbia la capacità se vuole di approfondire da solo i temi che più lo interessano. Un ruolo veramente fondamentale è poi svolto dalle esercitazioni in laboratorio, che formano parte integrante del Corso, giacché molti dei concetti presentati a lezione possono (e devono) essere verificati dagli studenti sui calcolatori messi a loro disposizione.

Il presente testo è una raccolta delle lezioni svolte durante il corso. Tuttavia le informazioni qui contenute sono certamente molte più di quelle presentate a lezione e richieste all'esame: si è cercato infatti di rendere questa raccolta una sorta di testo base di informatica per fisici, dove gli studenti potessero trovare le prime risposte al loro desiderio di conoscenza. Spesso infatti le nozioni di informatica che possono servire ad un laureato in Fisica sono sparse su più testi diversissimi fra loro per argomento e difficoltà. Una parte di questo materiale ha origine dall'esperienza diretta dell'Autore, specie per quanto concerne lo Unix; il resto è stato estratto da varie fonti, talora citate in bibliografia quando si è ritenuto utile segnalarle agli studenti. Nello scriverne il testo si è cercato di evitare un uso eccessivo di anglicismi sostituendoli quando possibile con corrette espressioni italiane, sfuggendo però l'eccesso opposto e preferendo un termine tecnico inglese ormai affermatosi nell'uso ad una storpiatura italiana. Certo questa raccolta è lungi dall'essere esaustiva, data l'ampiezza degli argomenti e la loro rapida evoluzione. Né può essere priva di errori (si dovrebbe dire *bachi* trattandosi di un testo di informatica), che pure ci si è sforzati di ridurre al minimo. La speranza dell'Autore rimane che gli studenti possano trovare utile questo testo come punto di partenza per addentrarsi in quelle conoscenze di informatica che risulteranno loro indispensabili nel mondo della ricerca come in quello del lavoro.

Mario Sitta, Alessandria, Dicembre 2000

La seconda versione di queste dispense, oltre alla correzione di alcuni errori sfuggiti ai precedenti controlli, contiene un quinto capitolo dedicato alle reti di calcolatori ed una più ampia appendice sui comandi dello Unix.

Sebbene sin dal passato Anno Accademico il Corso abbia mutato nome, divenendo Laboratorio di Calcolo I, i suoi fini rimangono gli stessi, e queste dispense continuano a costituire un'utile sussidio per gli Studenti alla preparazione dell'esame.

Mario Sitta, Alessandria, Ottobre 2002

Capitolo I

Numeri binari

"Can you do Addition?" the White Queen asked. "What's one and one and one and one and one and one and one and one and one and one?"
"I don't know," said Alice. "I lost count."
"She can't do Addition," the Red Queen interrupted.

Lewis Carroll, "Through the Looking-Glass"

§ 1.1 Numerazione binaria, ottale, esadecimale

Il sistema di numerazione comunemente adoperato è detto *numerazione decimale* poiché si basa sulle potenze di 10. Si tratta di un sistema *posizionale* in cui cioè il valore di ciascuna cifra dipende dalla sua posizione (contrariamente ad altri sistemi, come i numeri romani, in cui ogni cifra ha sempre lo stesso valore in qualunque posizione si trovi): per esempio il numero 3333 è formato da quattro cifre identiche ma ognuna di esse ha un valore diverso (mentre ad esempio nel numero romano XXX tutte le cifre hanno identico valore). Nel sistema decimale viene usato come *radice* o *base* il 10, quindi ogni cifra rappresenta quante volte una certa potenza di 10 è contenuta nel numero rappresentato: così ad esempio

$$5492 = 5 \times 10^3 + 4 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

e in generale

$$\begin{aligned} c_n c_{n-1} \ldots c_2 c_1 c_0 &= c_n \times 10^n + c_{n-1} \times 10^{n-1} + \ldots + c_2 \times 10^2 + c_1 \times 10^1 + c_0 \times 10^0 \\ &= \sum_{k=0}^{n} c_k \times 10^k. \end{aligned}$$

Per rappresentare i numeri decimali sono necessarie quindi dieci cifre.

Tuttavia 10 non è l'unica base possibile: qualunque numero intero dal 2 in poi può essere utilizzato per costruire un sistema di numerazione * . E in effetti il sistema più semplice è proprio il *sistema binario* che si ottiene utilizzando 2 come base: in questo caso infatti sono sufficienti due sole cifre, lo 0 e l'1, per rappresentare tutti i numeri. In un numero in base 2 ogni cifra rappresenta quante volte una certa potenza di 2 è contenuta nel numero dato: per esempio

$$10011 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \tag{1.1}$$

ovvero 19 in base 10. Per evitare ambiguità si usa apporre un pedice al numero per indicarne la base: così $10011_2 = 19_{10}$.

La numerazione binaria viene ampiamente utilizzata nei calcolatori elettronici poiché è comodo assegnare le cifre 0 e 1 ai due possibili stati di un circuito, aperto e chiuso; inoltre è possibile fare lo stesso tipo di assegnamento a condizioni logiche opposte, come vero e falso, acceso e spento, abilitato e disabilitato, e così via. L'adozione del sistema binario ha contribuito notevolmente allo sviluppo dei calcolatori elettronici proprio perché si adatta molto meglio del sistema decimale, il quale richiederebbe ben dieci stati distinti certo più complicati da ottenere.

In un numero binario ogni singola cifra 0 o 1 viene indicata come *bit* (per *binary digit*). I bit vengono numerati da destra a sinistra a partire da zero: in tal modo il numero che designa ciascun bit è anche l'esponente a cui va elevato il 2 per ottenere il valore di quella cifra all'interno del numero binario (per esempio il bit numero 3 vale $2^3 = 8$); il bit numero 0 all'estrema destra è il *bit meno significativo*, mentre quello all'estrema sinistra è il *bit più significativo*. Un *byte* è un gruppo di 8 bit che formano un insieme unitario. I byte costituiscono le unità fondamentali dal punto di vista del funzionamento concreto del calcolatore, perché i dati in memoria sia fissa sia volatile sono fisicamente raggruppati in celle di un byte l'una. Perciò il byte è universalmente impiegato come unità di misura della quantità di informazioni in un calcolatore. Vi sono $2^8 = 256$ modi diversi di disporre

* Teoricamente non è neppure necessario che la base sia intera, può essere utilizzato qualunque numero reale positivo diverso da 1; nella pratica però le basi intere sono le uniche impiegate.

i bit di un byte, quindi esso può rappresentare tutti i numeri decimali da 0 a 255. Multipli del byte comunemente utilizzati sono il *chilobyte*

$$1 \text{ kB} = 1024 \text{ byte} = 2^{10} \text{ byte}$$

il *megabyte*

$$1 \text{ MB} = 1024 \text{ kB} = 1\,048\,576 \text{ byte} = 2^{20} \text{ byte}$$

il *gigabyte*

$$1 \text{ GB} = 1024 \text{ MB} = 1\,073\,741\,824 \text{ byte} = 2^{30} \text{ byte}$$

e il *terabyte*

$$1 \text{ TB} = 1024 \text{ GB} = 1\,099\,511\,627\,776 \text{ byte} = 2^{40} \text{ byte} \qquad \ddagger$$

La conversione fra i due sistemi numerici è semplice. Per passare da un numero binario al suo equivalente decimale è sufficiente sommare le potenze di 2 che corrispondono alle cifre 1, come nell'Eq. (1.1). Per il passaggio inverso il metodo più semplice è effettuare delle divisioni per potenze decrescenti di 2: volendo convertire ad esempio il numero decimale 150 in binario si può procedere come segue

$$\begin{array}{rcrcll}
150 & : & 128 & = & 1 & \text{e resto } 22 \\
22 & : & 64 & = & 0 & \text{e resto } 22 \\
22 & : & 32 & = & 0 & \text{e resto } 22 \\
22 & : & 16 & = & 1 & \text{e resto } 6 \\
6 & : & 8 & = & 0 & \text{e resto } 6 \\
6 & : & 4 & = & 1 & \text{e resto } 2 \\
2 & : & 2 & = & 1 & \text{e resto } 0 \\
0 & : & 1 & = & 0 & \text{e resto } 0
\end{array}$$

il numero voluto si ottiene rileggendo la colonna dei risultati dall'alto verso il basso: in effetti è proprio

$$150 = 128 + 16 + 4 + 2 = 10010110_2$$

‡ Queste sono le definizioni standard usate anche dai sistemi operativi nei loro calcoli interni. Purtroppo però alcuni Costruttori usano una definizione "decimale" di questi multipli, ponendo ad esempio 1 kB = 1 000 byte, 1 MB = 1 000 000 byte, ecc., il che oltre a ingenerare confusione può portare all'acquisto di periferiche di capacità inferiore a quanto si possa ritenere (ciò capita sovente con i dischi rigidi).

Un secondo metodo consiste invece nell'operare ripetute divisioni per 2 ed esaminandone i resti: nell'esempio precedente

$$\begin{array}{rcrcrl}
150 & : & 2 & = & 75 & \text{e resto } 0 \\
75 & : & 2 & = & 37 & \text{e resto } 1 \\
37 & : & 2 & = & 18 & \text{e resto } 1 \\
18 & : & 2 & = & 9 & \text{e resto } 0 \\
9 & : & 2 & = & 4 & \text{e resto } 1 \\
4 & : & 2 & = & 2 & \text{e resto } 0 \\
2 & : & 2 & = & 1 & \text{e resto } 0 \\
1 & : & 2 & = & 0 & \text{e resto } 1
\end{array}$$

in questo caso il numero binario voluto si ricava rileggendo la colonna dei resti dal basso verso l'alto: così

$$150_{10} = 10010110_2$$

Un altro sistema numerico usato nei calcolatori è il *sistema ottale*, che ha come base 8 e utilizza le cifre da 0 a 7; in questo caso quindi ogni cifra indica una potenza di 8: ad esempio

$$243_8 = 2 \times 8^2 + 4 \times 8^1 + 3 \times 8^0 = 163_{10}$$

La conversione fra ottale e binario è immediata, in quanto ogni cifra ottale rappresenta tre cifre binarie: quindi dato un numero binario è sufficiente ragruppare a tre a tre le sue cifre a partire da destra per avere il corrispondente ottale, e viceversa basta esprimere in base 2 le cifre di un numero ottale per trasformarlo nel corrispondente binario: nell'esempio precedente di 150_{10} si ha

$$\underbrace{(0)10}_{2}\ \underbrace{010}_{2}\ \underbrace{110}_{6}$$

quindi $10010110_2 = 226_8$.

Un terzo sistema numerico molto usato è il *sistema esadecimale*, che ha come base 16 e necessita quindi di sedici cifre; poiché i numeri arabi ne hanno solo dieci, per convenzione unanime si usano le prime sei lettere maiuscole dell'alfabeto con questa corrispondenza

$$\begin{array}{lll}
\mathrm{A} = 10 & \mathrm{B} = 11 & \mathrm{C} = 12 \\
\mathrm{D} = 13 & \mathrm{E} = 14 & \mathrm{F} = 15
\end{array}$$

Anche questo è un sistema posizionale in cui ogni cifra indica quante volte compare una certa potenza di 16: per esempio

$$1\mathrm{A}9_{16} = 1 \times 16^2 + 10 \times 16^1 + 9 \times 16^0 = 425_{10}$$

(talora invece del pedice $_{16}$ si usa la lettera H per indicare un numero esadecimale, come 1A9H, oppure il prefisso 0x, come 0x1a9). Anche la conversione fra esadecimale e binario è immediata, basta raggruppare quattro a quattro i bit di un numero binario: una cifra esadecimale infatti rappresenta da sola il valore di mezzo byte (quest'ultimo prende il nome di *nibble*): usando ancora l'esempio di 150_{10} si ha

$$\underbrace{1001}_{9}\,\underbrace{0110}_{6}$$

quindi $10010110_2 = 96_{16}$. La conversione da decimale a ottale o esadecimale è invece più complessa: si può ricorrere come nel caso dei numeri binari a progressive divisioni per 8 o per 16. Va comunque notato che tutti i calcolatori posseggono dei programmi residenti preposti alla conversione di numeri fra basi diverse.

La notazione esadecimale si adatta molto bene alla forma in cui i dati sono rappresentati all'interno di un calcolatore, anche meglio del sistema ottale: infatti ogni cifra esadecimale esprime come si è detto il valore di mezzo byte, e un byte può essere facilmente scritto con due cifre esadecimali, da 00H fino a FFH (pari a 255 decimale).

Finora si è implicitamente convenuto si trattasse solo di numeri interi, ma in maniera immediata si può estendere quanto detto ai numeri con la virgola: ogni cifra dopo la virgola indica una potenza negativa della base, per esempio

$$81.53_{10} = 8 \times 10^1 + 1 \times 10^0 + 5 \times 10^{-1} + 3 \times 10^{-2}$$

$$1010.101_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$53.02_8 = 5 \times 8^1 + 3 \times 8^0 + 0 \times 8^{-1} + 2 \times 8^{-2}$$

$$12B.D3_{16} = 1 \times 16^2 + 2 \times 16^1 + 11 \times 16^0 + 13 \times 16^{-1} + 3 \times 16^{-2}$$

(si noti che è prassi comune seguire la convenzione anglosassone per cui la virgola è rappresentata da un punto).

§ 1.2 Operazioni con numeri binari

1.2.1 Operazioni algebriche

In questi sistemi numerici le quattro operazioni aritmetiche assumono la stessa forma che hanno nella numerazioni decimale, con tanto di riporti, prestiti e resti, basta ricordare che analogamente a $9+1=10_{10}$ si ha ora $1+1=10_2$, $7+1=10_8$ e $\mathrm{F}+1=10_{16}$. Si considerino come esempio le seguenti operazioni fra gli *stessi* operandi espresse nei sistemi esposti in precedenza

$$\begin{array}{rl@{\qquad}rl@{\qquad}rl@{\qquad}rl}
\text{base 10} & & \text{base 2} & & \text{base 8} & & \text{base 16} & \\
 & & & & & & & \\
18 & + & 10010 & + & 22 & + & 12 & + \\
7 & = & 111 & = & 7 & = & 7 & = \\
\cline{1-1}\cline{3-3}\cline{5-5}\cline{7-7}
25 & & 11001 & & 31 & & 19 &
\end{array}$$

$$\begin{array}{rl@{\qquad}rl@{\qquad}rl@{\qquad}rl}
54 & - & 110110 & - & 66 & - & 36 & - \\
15 & = & 1111 & = & 17 & = & \mathrm{F} & = \\
\cline{1-1}\cline{3-3}\cline{5-5}\cline{7-7}
39 & & 100111 & & 47 & & 27 &
\end{array}$$

$$\begin{array}{rl@{\qquad}rl@{\qquad}rl@{\qquad}rl}
72 & \times & 1001000 & \times & 110 & \times & 48 & \times \\
19 & = & 10011 & = & 23 & = & 13 & = \\
\cline{1-1}\cline{3-3}\cline{5-5}\cline{7-7}
648 & & 1001000 & & 330 & & \mathrm{D8} & \\
72- & & 1001000- & & 220- & & 48- & \\
 & & 1001000 & & & & & \\
\cline{1-1}\cline{3-3}\cline{5-5}\cline{7-7}
1368 & & 10101011000 & & 2530 & & 558 &
\end{array}$$

in modo analogo possono essere trattate pure le divisioni (anche se in realtà i calcolatori usano prima valutare il reciproco del divisore per poi moltiplicarlo per il dividendo).

Nel sistema decimale moltiplicare o dividere per 10 è equivalente a spostare tutte le cifre di una posizione verso sinistra o verso destra, aggiungendo uno zero se necessario

$$175 \times 10 = 1750$$

Ma anche nel sistema binario moltiplicare o dividere per $2 = 10_2$ è equivalente a spostare di una posizione le cifre del moltiplicando

$$10011_2 \times 10_2 = 100110_2$$

(cioè $19_{10} \times 2_{10} = 38_{10}$). Lo stesso vale nel sistema ottale ed esadecimale moltiplicando e dividendo rispettivamente per $8 = 10_8$ e $16 = 10_{16}$ (e in generale in qualunque sistema moltiplicando o dividendo un numero per una potenza della sua base).

1.2.2 Operazioni logiche

Accanto alle operazioni aritmetiche esistono delle *operazioni logiche*; esse pur potendo essere effettuate fra numeri in qualsiasi base hanno significato solo attraverso i loro equivalenti binari. Le operazioni principali sono

- il *prodotto logico* o *AND* di due numeri binari, che è un numero binario avente una cifra 1 in una data posizione solo se entrambi gli operandi hanno un 1 in quella posizione: ad esempio

$$0101\,0111 \quad \mathrm{AND} \quad 0100\,0101 = 0100\,0101$$

- la *somma logica* o *OR* di due numeri binari, che è un numero binario avente una cifra 1 in una data posizione se almeno uno dei due operandi ha un 1 in quella posizione: per esempio

$$0101\,0111 \quad \mathrm{OR} \quad 0010\,0101 = 0111\,0111$$

  una variante è l'*OR esclusivo* o *XOR*, in cui il risultato ha una cifra 1 in una data posizione se uno dei due operandi ha un 1 in quella posizione ma non entrambi:

$$0101\,0111 \quad \mathrm{XOR} \quad 0010\,0101 = 0111\,0010$$

- la *negazione logica* o *NOT* di un numero binario, che è un numero binario avente un 1 nelle posizioni in cui l'operando ha uno 0 e viceversa

$$\mathrm{NOT} \quad 0101\,0111 = 1010\,1000$$

  (questa operazione viene anche indicata come *complemento a 1*).

1.2.3 Numeri negativi

Finora sono stati presi in considerazione solo numeri positivi, ma con lo stesso sistema si possono rappresentare anche i numeri negativi: è sufficiente utilizzare il bit più significativo per indicarne il segno. Negli esempi seguenti si utilizzeranno numeri di un byte, ma ovviamente tutti i concetti esposti valgono per numeri binari di qualsiasi estensione.

Il sistema più semplice consiste nell'usare il bit più significativo per indicare il segno (0 per il $+$ e 1 per il $-$) e i restanti bit per il valore assoluto del numero: con 8 bit si possono così rappresentare tutti i numeri interi fra -127 e $+127$:

Numeri positivi		Numeri negativi	
Base 10	Base 2	Base 10	Base 2
1	0000 0001	-1	1000 0001
2	0000 0010	-2	1000 0010
3	0000 0011	-3	1000 0011
4	0000 0100	-4	1000 0100
5	0000 0101	-5	1000 0101
$\vdots$	$\vdots$	$\vdots$	$\vdots$

In questo caso tuttavia è necessaria una regola speciale per sommare numeri negativi: infatti con questa notazione

$$+1 + (-1) = 0000\,0001 + 1000\,0001 = 1000\,0010 = -2 \text{ e non } 0\ !$$

Sarebbe certamente più comoda una rappresentazione dei numeri positivi e negativi nella quale valgano le solite regole algebriche e non sia necessario invece prima esaminare il segno degli operandi per poi decidere come svolgere l'operazione. Si consideri allora la seguente serie di sottrazioni successive

$$\begin{aligned}
3 - 1 &= 2 \Longrightarrow 0000\,0011 - 0000\,0001 = 0000\,0010 \\
2 - 1 &= 1 \Longrightarrow 0000\,0010 - 0000\,0001 = 0000\,0001 \\
1 - 1 &= 0 \Longrightarrow 0000\,0001 - 0000\,0001 = 0000\,0000 \\
0 - 1 &= -1 \Longrightarrow 0000\,0000 - 0000\,0001 = 1111\,1111 \\
-1 - 1 &= -2 \Longrightarrow 1111\,1111 - 0000\,0001 = 1111\,1110
\end{aligned}$$

quindi da questo punto di vista la rappresentazione più "naturale" di -1 è 1111 1111, di -2 è 1111 1110 e così via. Si noti che

$$+1 + (-1) = 0000\,0001 + 1111\,1111 = 0000\,0000 = 0$$

e per esempio

$$19 + (-2) = 0001\,0011 + 1111\,1110 = 0001\,0001 = 17$$

Dato un numero positivo N il corrispondente negativo $-N$ si ottiene aggiungendo 1 al complemento a uno del numero $\tilde{N}$:

$$-N = \tilde{N} + 1$$

ad esempio

$$5_{10} = 0000\,0101 \Longrightarrow \tilde{5} = 1111\,1010 \Longrightarrow -5 = 1111\,1011$$

questa operazione è detta *complemento a 2*. E' possibile verificare che con questa rappresentazione tutte le operazioni algebriche conservano le stesse regole valide nell'usuale notazione decimale.

Entrambe le rappresentazioni dei numeri negativi presentano un difetto. La prima è simmetrica, nel senso che comprende tutti i numeri tra -127 e $+127$ e quindi ogni numero positivo ha il corrispondente negativo; però in essa vi sono due rappresentazioni dello zero, $+0 = 0000\,0000$ e $-0 = 1000\,0000$. Al contrario la seconda possiede una sola rappresentazione dello 0, ma è asimmetrica in quanto esprime tutti i numeri compresi fra -128 e $+127$ e quindi c'è un numero negativo senza controparte positiva. Uno dei due difetti è inevitabile: infatti si vorrebbe un sistema che contenga tanti numeri positivi quanti negativi e possieda una sola rappresentazione dello zero; ma un insieme con m numeri positivi, m negativi ed uno zero ha un numero dispari di elementi $(2m + 1)$, laddove con n bit si ottengono sempre insiemi con un numero pari di elementi $(2n)$: per un qualsiasi n ci sarà sempre o una disposizione di bit di troppo o una di meno di quante ne servirebbero, qualunque sia la notazione scelta. Questa disposizione in più può essere usata per indicare un numero negativo o positivo molto grande, o per dare una seconda rappresentazione dello 0, o per qualcos'altro, ma sarà sempre una scomodità.

§ 1.3 Rappresentazione interna dei numeri

E' evidente che i numeri come ogni altra informazione sono immagazzinati nel calcolatore in byte. I numeri interi vengono registrati in base alla loro rappresentazione binaria; a seconda del valore massimo che si vuole gestire essi possono occupare un solo byte oppure due (in tal caso l'intero prende nome di *parola* o *word*), quattro (*integer* o *long word*) e in taluni calcolatori anche otto (*long integer*).

I numeri reali hanno invece una rappresentazione più complessa. Serve infatti un sistema in cui l'intervallo di numeri esprimibili sia indipendente dal numero di cifre significative. Il metodo migliore per separare l'estensione dei numeri dalla loro precisione è la usuale notazione scientifica, che in base 10 permette di esprimere ogni numero come

$$n = f \times 10^e$$

dove f è detta *frazione* o *mantissa* e e è un intero positivo o negativo detto *esponente*; in questo caso l'intervallo di numeri rappresentabili è determinata dal numero di cifre dell'esponente mentre la loro precisione dal numero di cifre della mantissa.

La versione per calcolatore della notazione scientifica è il formato *a virgola mobile* (*floating point format*), in cui si usa 2 come base e quindi ogni numero può essere scritto come

$$n = F \times 2^E$$

Per rappresentare un numero in virgola mobile sono necessarî un esponente E, una mantissa F, un segno per l'esponente ed un segno per la mantissa; per quest'ultimo si usa al solito il bit più significativo mentre il segno dell'esponente non è esplicitamente espresso in quanto per convenzione dall'esponente va sottratto un numero fisso; la mantissa poi è un numero frazionario binario. La normalizzazione della mantissa di un numero è però ambigua giacché lo spostamento di una posizione delle sue cifre può essere compensato da una variazione di una unità dell'esponente: per esempio tutte le forme seguenti sono adatte a rappresentare il numero 32

$$2^8 \times 0.00100$$

$$2^7 \times 0.01000$$

$$2^6 \times 0.10000$$

$$2^5 \times 1.00000$$

Ma l'ultima è più vantaggiosa perchè conserva il numero massimo di bit significativi: per esprimere ad esempio il numero 33 solo l'ultima rappresentazione conserva l'accuratezza dell'ultimo bit. Quindi la forma della mantissa con il bit più significativo a 1 (*mantissa normalizzata*) è preferibile in quanto massimizza le cifre significative e quindi l'accuratezza dei numeri esprimibili. L'unica ragione per usare una mantissa con il bit più significativo non a 1 (*mantissa denormalizzata*) è che a volte potrebbe non essere disponibile un esponente sufficiente per esprimere numeri molto piccoli: cioè sarebbe possibile rappresentare numeri più piccoli di quanto permetta l'intervallo dell'esponente se si usassero mantisse denormalizzate. Tuttavia siccome è assai improbabile che numeri così piccoli siano significativi nei calcoli e comunque gli esponenti hanno intervalli di variabilità abbastanza ampi, l'IEEE ha deciso di adottare lo standard della mantissa normalizzata, in cui cioè il bit più significativo è posto sempre a 1 tranne quando l'esponente E è 0, nel cui caso la mantissa è denormalizzata per poter rappresentare anche lo zero. Ciò consente di *sottintendere* questo bit e quindi di guadagnare un ulteriore bit di accuratezza.

Solitamente per rappresentare un numero in virgola mobile si adoperano quattro byte (*numeri in singola precisione*): in questo caso il bit più significativo è il segno della mantissa, seguono 8 bit per l'esponente (rappresentato come eccesso a 127 in modo da non necessitare di un bit di segno), e i restanti 23 bit sono dedicati a rappresentare la mantissa

31	30 29 28 27 26 25 24 23	22 21 20 19 ... 3 2 1 0
S	Esponente	Mantissa

L'esponente è lungo un byte ma è spostato a destra di un bit per fare spazio al bit di segno e quindi non corrisponde al byte più a sinistra; la mantissa è lunga un bit in meno di tre byte, ma tenendo conto del bit a 1 sottinteso è di fatto lunga proprio tre byte. Pertanto ogni numero in virgola mobile è scritto come

$$(-1)^S \times 1.M_{22}M_{21}M_{20}M_{19}\ldots M_3M_2M_1 \times 2^{E-127}$$

dove S è il bit di segno, E il valore dell'esponente e M_i i bit della mantissa. Ad esempio, +1 viene rappresentato come

$$\begin{aligned} & (-1)^0 \times 1.000\ldots000 \times 2^{127-127} \\ = \; & 0011 \;\; 1111 \;\; 1000 \;\; 0000 \;\; 0000 \;\; 0000 \;\; 0000 \;\; 0000 \\ = \; & 3 \qquad F \qquad 8 \qquad 0 \qquad 0 \qquad 0 \qquad 0 \qquad 0 \end{aligned}$$

infatti il primo bit è il segno, gli otto bit seguenti l'esponente aumentato di $127 = 01111111$ e quindi seguono i 22 bit della mantissa, tutti a 0 essendo il primo 1 sottinteso. Analogamente si scrivono i primi numeri

$$+1 = 1.000\ldots000 \times 2^0 = \text{3F800000} = 0\,\underbrace{01111111}_{\text{esp}}\,00000\ldots000$$

$$+2 = 1.000\ldots000 \times 2^1 = 40000000 = 0\,\underbrace{10000000}_{\text{esp}}\,00000\ldots000$$

$$+3 = 1.100\ldots000 \times 2^1 = 40400000 = 0\,\underbrace{10000000}_{\text{esp}}\,10000\ldots000$$

$$+4 = 1.000\ldots000 \times 2^2 = 40800000 = 0\,\underbrace{10000001}_{\text{esp}}\,00000\ldots000$$

$$+5 = 1.010\ldots000 \times 2^2 = 40A00000 = 0\,\underbrace{10000001}_{\text{esp}}\,01000\ldots000$$

In questo modo si possono rappresentare numeri con modulo da circa $1.18 \cdot 10^{-38}$ fino a circa $3.40 \cdot 10^{38}$ (e si può scendere fino a circa $1.40 \cdot 10^{-45}$ se si ammettono anche mantisse denormalizzate); la precisione è di una parte su 2^{23} cioè circa 7 cifre decimali. Quando i calcoli richiedono una accuratezza maggiore, si possono usare otto byte invece di quattro (*numeri in doppia precisione*): in questo caso il bit 63 è il segno, i bit da 52 a 62 sono usati per l'esponente (rappresentato come eccesso a 1024) e i bit da 0 a 51 per la mantissa; il modulo dei numeri rappresentabili varia da circa $2.23 \cdot 10^{-308}$ a circa $1.80 \cdot 10^{308}$, mentre la precisione è di una parte su 2^{52} cioè circa 15 cifre decimali.

I numeri in virgola mobile possono essere usati come modello per i numeri reali, ma vi sono delle differenze concettuali importanti. La linea dei numeri reali può essere suddivisa in sette regioni diverse

1. numeri negativi minori di $-3.40 \cdot 10^{38}$
2. numeri negativi compresi fra $-3.40 \cdot 10^{38}$ e $-1.18 \cdot 10^{-38}$
3. numeri negativi maggiori di $-1.18 \cdot 10^{-38}$
4. lo zero
5. numeri positivi inferiori a $1.18 \cdot 10^{-38}$
6. numeri positivi compresi fra $1.18 \cdot 10^{-38}$ e $3.40 \cdot 10^{38}$
7. numeri positivi superiori a $3.40 \cdot 10^{38}$

I numeri in virgola mobile non possono essere usati per rappresentare un numero appartenente alle regioni 1, 3, 5 o 7, giacché questi numeri escono dall'intervallo di valori gestibili. Se il risultato di una operazione matematica è un valore appartenente alle zone 1 o 7, ad esempio $-5 \cdot 10^{40}$ oppure $7 \cdot 10^{56}$, si genera un *errore di overflow* cioè di superamento del massimo valore gestibile e il risultato ottenuto sarà inesatto. Analogamente se il risultato di una operazione appartiene alle regioni 3 o 5, per esempio $-3 \cdot 10^{-42}$, si genera un *errore di underflow* cioè di superamento del valore assoluto minimo gestibile; questo errore però è meno grave, in quanto spesso il valore 0 è una sufficiente approssimazione di valori così piccoli. Tutto ciò è inevitabile data la natura di qualsiasi rappresentazione dei numeri con un numero finito di cifre, quale necessariamente deve essere in un calcolatore.

Un'altra importante differenza fra i numeri reali e i numeri in virgola mobile è la loro densità. Fra due qualsiasi numeri reali si trova sempre almeno un altro numero reale (in realtà ve ne sono infiniti): i numeri reali infatti formano un continuo. Al contrario i numeri in virgola mobile non formano un continuo: vi è un numero finito di numeri esprimibili in questa rappresentazioni (in realtà in qualsiasi rappresentazione), ed è alquanto probabile che il risultato di una operazione sia un numero non rappresentabile. Ad esempio il risultato di $1 : 3$ non può essere espresso *esattamente* in questo sistema. Se un numero non ha rappresentazione, la cosa migliore è usare quello che più gli si avvicina: questo processo è noto come *arrotondamento*. Non solo: la spaziatura fra due numeri consecutivi rappresentabili non è costante lungo tutta la retta dei reali. Per esempio la separazione fra $1.1111\ldots1110 \times 2^{127}$ e $1.1111\ldots1111 \times 2^{127}$ è enormemente maggiore di quella fra $1.1111\ldots1110\times2^{-127}$ e $1.1111\ldots1111\times2^{-127}$. Quando tuttavia tale separazione è espressa come frazione del numero, si vede che non vi è variazione lungo tutte le regioni 2 e 6, ovvero l'errore *relativo* introdotto dalla rappresentazione è costante.

Bibliografia

W. R. Gibbs, *Computation in Modern Physics*, ed. World Scientific, 1991

A. S. Tanenbaum, *Structured Computer Organization*, ed. Prentice–Hall, 1984

The binary system is really fun,
For with it strange things can be done.
Two as you know
Is a one and an oh,
And five is one hundred and one.

Anon. (citato in M. Gardner "Knotted Doughnuts")

Capitolo II

Generalità sui calcolatori

I would like these calculations were made by steam!

Charles Babbage

§ 2.1 Cenni storici

I tentativi di realizzare strumenti per velocizzare e automatizzare le operazioni di calcolo risalgono alla più remota antichità: si pensi solo agli abachi e ai pallottolieri in uso presso babilonesi, cinesi, greci, romani, nel Medioevo, e ancora oggi presso alcuni popoli orientali.

Le prime *macchine* calcolatrici però possono essere considerate le addizionatrici costruite da B. Pascal e G. W. von Leibniz alla metà del XVII secolo: esse operavano mediante ruote dentate azionate manualmente ed in grado di risolvere automaticamente il problema dei riporti.

Nel XIX secolo C. Babbage progettò diversi dispositivi per il calcolo automatico. In particolare realizzò parzialmente quella che egli chiamò la *macchina alle differenze* (*Difference Engine*, un modello funzionante della quale è stato costruito alcuni anni fa presso lo Science Museum di Londra basandosi sui progetti originali di Babbage). Esasperato per l'imprecisione delle tavole numeriche pubblicate all'epoca, immaginava che i calcoli potessero essere eseguiti da una macchina. Quella che progettò costituisce la prima macchina da calcolo automatica: una volta avviata non richiede un intervento esterno per portare a termine le operazioni. Essa consente di tabulare polinomi di settimo grado fino a 31 cifre, un risultato eccezionale anche per i sistemi attuali. Tuttavia ha un difetto: può eseguire un solo compito fisso. Così Babbage cercò di progettare e sviluppare un dispositivo più sofisticato, la *macchina analitica* (*Analytical Engine*): questa sarebbe dovuta essere una macchina da calcolo programmabile di uso generale. Dotata di un repertorio di operazioni aritmetiche che potevano essere effettuate in qualsiasi ordine, era anche in grado di seguire vie alternative a seconda del risultato di un calcolo, il che le permetteva di eseguire funzioni complesse. La sua struttura interna prevedeva un "mulino" ed un "magazzino" separati, equivalenti alla unità di elaborazione e alla memoria dei calcolatori attuali, separazione che è una caratteristica dei calcolatori elettronici a partire dalla metà degli anni Quaranta. Alla macchina analitica era possibile fornire i dati iniziali ed il programma da eseguire mediante schede perforate, una tecnica già usata nel telaio Jacquard per produrre tessuti con disegni diversi. Babbage prevedeva che la macchina analitica accettasse dati numerici fino a 50 cifre e fornisse risultati anche di 100 cifre, che potevano essere stampati, riportati in grafico o su altre schede perforate. Purtroppo però essa rimase solo una serie di progetti, che Babbage portò avanti per quasi 40 anni fino alla sua morte, senza mai tentare di costruire una versione a grandezza naturale, come invece avvenne per la macchina alle differenze, di cui esistono prototipi parziali dell'epoca ancora funzionanti.

Tra la fine del XIX secolo e l'inizio del XX la tecnologia meccanica raggiunse uno sviluppo tale da permettere la realizzazione di calcolatrici da tavolo in grado di eseguire con rapidità le operazioni aritmetiche. Per risolvere i problemi di conteggio legati al censimento della popolazione degli Stati Uniti, nel 1890 H. Hollerith brevettò una *tabulatrice* a schede perforate, nella quale i dati venivano introdotti mediante schede di cartone su cui essi erano codificati con opportune combinazioni di fori. In questo tipo di macchine le operazioni erano eseguite da organi elettromeccanici, che seguivano schemi di calcolo determinati variando le connessioni di pannelli elettrici.

Durante la Seconda Guerra Mondiale i calcolatori elettromeccanici raggiunsero il culmine del loro sviluppo. Il massimo perfezionamento è rappresentato da un calcolatore analogico, l'*analizzatore differenziale* di Vannevar Bush, un matematico del MIT. Interamente meccanico (soltanto i motori che l'azionavano erano elettrici), con esso era possibile compilare una tabella di tiro per l'artiglieria in soli 15 minuti, contro le 20 ore necessarie ad un operatore umano. Poiché però ne esistevano soltanto due esemplari, si cominciò a progettare e realizzare una versione elettronica. Il primo calcolatore automatico elettronico fu l'ENIAC (*Electronic Numerical Integrator And Calculator*), costruito presso l'Università della Pennsylvania ed usato ancora dall'esercito statunitense per calcoli balistici; pesante 30 tonnellate e costituito da più di 18000 valvole che dissipavano 150 kW di potenza, in esso il programma di calcolo era ancora determinato dal cablaggio elettrico fra i diversi componenti. Solo alcuni anni più tardi furono realizzati i primi calcolatori, l'EDVAC (*Electronic Discrete Variable Automatic Computer*) negli Stati Uniti e l'EDSAC (*Electronic Delay Storage Automatic Calculator*) in Gran Bretagna, nei quali le istruzioni per l'elaborazione erano immagazzinate nella memoria stessa del calcolatore, costituita da linee di ritardo acustiche a mercurio, in una forma strutturalmente simile ai dati da elaborare. Questo concetto nuovo, cioè che il programma da eseguire possa essere memorizzato nel calcolatore stesso e in modo analogo ai dati, è di fondamentale importanza per i calcolatori moderni.

Agli inizi degli anni Cinquanta nacquero le prime industrie di calcolatori. I primi furono a valvole, e fra questi fu un successo l'UNIVAC (*Universal Automatic Computer*), che per primo utilizzava anche nastri magnetici. Invece l'IBM 704 fu il primo dotato di memorie a nucleo di ferrite, che permettevano una alta velocità di accesso e una grande capacità di memorizzazione. Con questa macchina poi vide la luce pure il primo vero linguaggio di programmazione, il FORTRAN (*Formula Translation*). Negli anni '60 si incominciò ad impiegare circuiti a semiconduttori (il primo ad utilizzare transistor fu l'IBM 7090), il che permise una notevole riduzione delle dimensioni e della potenza utilizzata; alla fine di quegli anni si cominciò ad impiegare anche la tecnologia dei ciruiti integrati. Ebbe così inizio quel vertiginoso sviluppo della tecnologia dei calcolatori elettronici che prosegue tuttora.

§ 2.2 Calcolatori analogici e numerici

I calcolatori possono essere suddivisi in due grandi categorie, i *calcolatori analogici* e i *calcolatori numerici* o *digitali*. Storicamente i calcolatori analogici furono i primi a raggiungere prestazioni elevate e rilevanti; tuttavia la nascita ed il rapidissimo sviluppo dei calcolatori numerici ne hanno ridotto notevolmente l'uso, riservandolo ad ambiti molto specialistici.

In entrambi i casi si tratta di macchine programmabili per l'esecuzione automatica di operazioni matematiche; la differenza sostanziale sta nel fatto che i calcolatori analogici operano su variabili continue (generalmente costituite da misure di grandezze fisiche), mentre i calcolatori numerici operano su quantità discrete trattate in forma codificata (solitamente numerica).

I calcolatori analogici sono dispositivi che analizzano e risolvono un problema facendo uso di un sistema che sia "analogo", cioè avente la stessa soluzione formale, ma che sia più semplice da gestire. Tale sistema non deve necessariamente essere elettrico o elettronico: esistono infatti calcolatori analogici meccanici, pneumatici, oleodinamici, elettromeccanici, e così via*; tutti però hanno in comune la capacità di poter essere programmati per simulare il comportamento di un insieme più o meno ampio di problemi, riconducendo le grandezze proprie dei sistemi simulati ad altre grandezze più facilmente accessibili all'utente.

Ad esempio, si consideri il sistema fisico rappresentato in Fig. 2.1a: una massa m è vincolata ad una molla di costante elastica k e si muove in un fluido viscoso di costante η. Come è noto, la forza $f(t)$ che agisce ad ogni istante sulla massa m è data da

$$f(t) = m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} + \eta\frac{\mathrm{d}x}{\mathrm{d}t} + kx = m\frac{\mathrm{d}v}{\mathrm{d}t} + \eta v + k\int v\ dt \tag{2.1}$$

dove x è lo spostamento verticale e v la velocità della massa. Si consideri ora il circuito di Fig. 2.1b: la caduta di potenziale totale è data da

$$V(t) = L\frac{\mathrm{d}i}{\mathrm{d}t} + Ri + \frac{Q}{C} = L\frac{\mathrm{d}i}{\mathrm{d}t} + Ri + \frac{1}{C}\int i\ dt \tag{2.2}$$

dove i è la corrente che circola nel circuito. Ora se si confrontano i termini delle equazioni (2.1) e (2.2) si vede che sono identici nella forma: pertanto pure le soluzioni saranno formalmente identiche. Se pertanto si fa corrispondere la tensione $V(t)$ alla forza $f(t)$, ne segue che alla corrente $i(t)$ corrisponde la velocità $v(t)$, all'induttanza L la massa m, alla resistenza R il coefficiente di viscosità η, e all'inverso della capacità $1/C$ la costante elastica k. Tuttavia il circuito elettrico è molto più facile da gestire del sistema meccanico: per un operatore è infatti decisamente più semplice per esempio variare una resistenza con un potenziometro che non cambiare il fluido in cui è immersa la massa con uno di

* Il regolo calcolatore è un esempio di calcolatore analogico meccanico, in cui la somma di logaritmi è riportata alla somma fra segmenti.

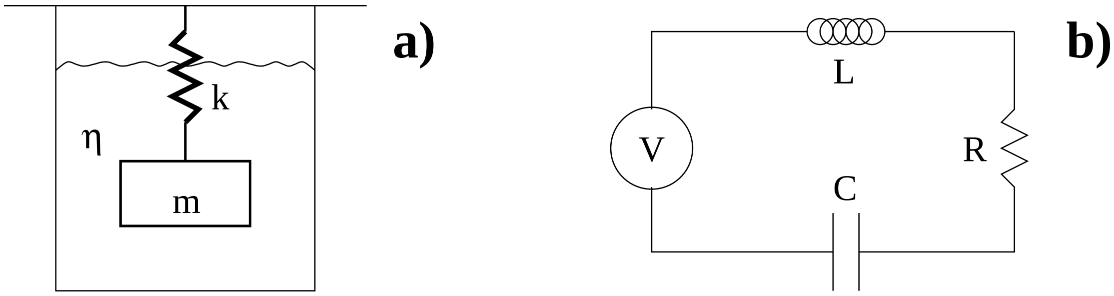

Fig. 2.1 *Due problemi fisici aventi la medesima soluzione formale*

costante η diversa. Quindi il circuito di Fig. 2.1b può essere usato molto agevolmente come calcolatore analogico per risolvere l'equazione del moto della massa m nel problema di Fig. 2.1a: in maniera molto semplice, leggendo un indicatore di corrente, è possibile conoscere la velocità della massa per un ampio intervallo dei parametri iniziali m, k, η. (E' da notare che a rigor di logica anche il meccanismo di Fig. 2.1a può essere considerato un calcolatore analogico per risolvere il problema del circuito elettrico di Fig. 2.1b; ma dal punto di vista pratico tale calcolatore sarebbe molto più complicato del problema di partenza.)

In generale in un calcolatore analogico l'andamento delle variabili che interessano non viene ricavato mediante un algoritmo procedurale, ma è prodotto direttamente e con continuità dal funzionamento stesso del simulatore. Come conseguenza risultano disponibili, al contempo e senza aggravio di complessità, anche variabili intermedie che possono interessare: quindi un calcolatore analogico è dotato della importantissima proprietà di operare rigorosamente in parallelo, essendo capace di far evolvere tutte le variabili in assoluta contemporaneità.

Caratteristica dei calcolatori analogici è la possibilità, almeno in teoria, di assumere un numero infinito di stati interni. Questa possibilità di trovarsi in infiniti stati tuttavia ne rappresenta anche un limite per quanto concerne la precisione, la quale può essere migliorata solo aumentando a dismisura le dimensioni del calcolatore, il che però si traduce nella realtà in un aumento delle imprecisioni costruttive, rendendo vano lo sforzo. Un ulteriore svantaggio consiste nella difficoltà di programmare un calcolatore analogico per eseguire compiti diversi.

Viceversa un calcolatore numerico risolve un problema se posto in una forma codificata che esso è in grado di interpretare. Anche questo non necessariamente deve essere elettrico o elettronico*, sebbene l'attuale sviluppo della microelettronica abbia reso i calcolatori elettronici di gran lunga più potenti e versatili.

Affinché un calcolatore numerico sia in grado di affrontare un problema o di eseguire un programma, questi devono prima essere tradotti in simboli o istruzioni semplici facenti parte del "vocabolario" che il calcolatore è in grado di comprendere. Tale traduzione può essere effettuata da un operatore umano, da programmi ausiliari già contenuti nel calcolatore, o da entrambi in gradi diversi. Il calcolatore numerico quindi deve essere dotato di dispositivi atti a riceve dati e istruzioni in ingresso e a presentare i risultati in uscita.

Il calcolatore numerico porta a termine i calcoli effettuando una successione di operazioni sui dati in base alle istruzioni ricevute. Le elaborazioni realizzate, per quanto complesse possano essere, risultano sempre composte da una serie di operazioni elementari di pochi tipi, come operazioni aritmetiche (somme, sottrazioni, ...), trasferimenti di informazioni fra zone diverse del calcolatore, confronti tra dati, operazioni di ingresso e uscita, e così via. Il calcolatore numerico inoltre è in grado, se gli è stato specificato, di prendere decisioni diverse in base ai dati a disposizione e quindi di portare a termine le operazioni di calcolo in modo differente a seconda di come questo evolve.

Contrariamente al calcolatore analogico un calcolatore numerico può assumere solo un numero finito di stati, corrispondenti a tutte le possibili combinazioni di stati dei suoi componenti interni. In esso poi il risultato finale come tutti i passaggi intermedi sono ottenuti applicando algoritmi procedurali ai dati. Il calcolatore numerico può avere diverse rappresentazioni interne delle grandezze numeriche, ma tutte sono accomunate dall'avere una risoluzione finita: ogni calcolatore numerico opera su variabili discrete e non continue. Nello svolgimento di operazioni matematiche necessariamente vengono introdotti degli errori dovuti agli inevitabili arrotondamenti; occorre essere consapevoli dell'entità di tali errori, perché possono essere significativi nella risoluzione del problema in esame. Il prin-

* Un abaco è un esempio di un semplice calcolatore numerico meccanico.

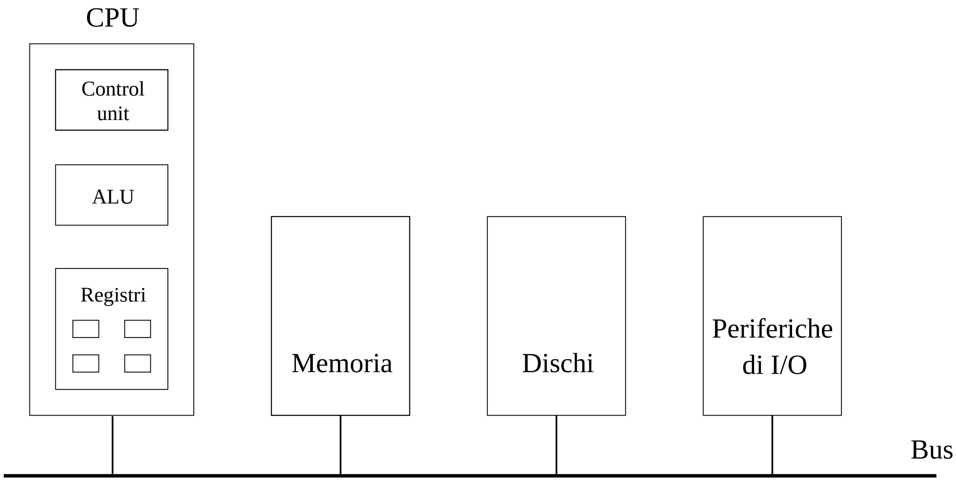

Fig. 2.2 *Disposizione schematica dei varî componenti di un calcolatore elettronico*

cipale vantaggio di un calcolatore numerico resta però la grande facilità nel programmarlo per eseguire compiti assai diversi fra loro.

D'ora in poi verranno trattati in dettaglio soltanto i calcolatori numerici elettronici.

§ 2.3 Le componenti di un calcolatore elettronico

Si è detto che un calcolatore digitale effettua una serie di operazioni su un insieme di dati al fine di risolvere un problema. Quindi sarà costituito da tre elementi essenziali: un elemento che effettua tali operazioni, un elemento atto a conservare, in modo temporaneo o permanente, i dati iniziali e i risultati finali, nonché se necessario i risultati parziali dei passaggi intermedi, ed un elemento che consenta la comunicazione fra i primi due elementi e l'operatore umano (ovvero altri calcolatori o strumenti di misura) al fine di poter inserire la serie di operazioni da svolgere e i dati su cui operare ed estrarre i risultati. Il primo elemento è il *processore* del calcolatore, il secondo si identifica nella *memoria volatile* e nelle *memorie di massa*, il terzo è formato dalle varie *periferiche di input/output*. In Fig. 2.2 è mostrata in modo schematico la connessione fra questi elementi.

2.3.1 Il processore

Il *processore** (o *CPU* per *Central Processing Unit*) è il "cervello" del calcolatore

* Spesso si usa il termine *microprocessore* in quanto costituito da un singolo chip elettronico.

elettronico. Il suo compito è di eseguire i programmi contenuti in memoria prelevando da essa le relative istruzioni, decodificandole ed eseguendole una dopo l'altra.

L'insieme delle istruzioni che un processore è in grado di eseguire costituiscono il suo *linguaggio macchina* o *linguaggio assembly* (pur riferendosi alla stessa cosa, si suole chiamarlo linguaggio assembly quando ci si riferisce al linguaggio impiegato dal programmatore per scrivere un programma, mentre con linguaggio macchina si intende lo stesso quando viene usato dal processore). Tutti i programmi che possono essere eseguiti direttamente dal processore sono scritti in questo linguaggio, che costituisce il linguaggio base del calcolatore, l'unico che esso sia in grado di comprendere; programmi scritti in altri linguaggi più "evoluti" devono prima essere "tradotti" in linguaggio macchina per poter essere eseguiti. Si tratta infatti di istruzioni molto semplici, come sommare due numeri, spostare un dato da o verso la memoria, confrontare due numeri o cambiare l'ordine di esecuzione. Per questo istruzioni complesse, quali si trovano nei linguaggi più alti, devono prima essere scomposti in una opportuna serie di istruzioni semplici. Il loro numero varia moltissimo a seconda del microprocessore impiegato, da poco più di 20 a oltre 300. Non necessariamente un ampio insieme di istruzioni in linguaggio macchina è preferibile ad un insieme ristretto, anzi sovente è vero il contrario, perché un insieme ampio spesso significa che le istruzioni non sono molto generali. I compilatori per linguaggi di alto livello generalmente danno prestazioni migliori su processori con un piccolo insieme di istruzioni ben scelte (detti per questo processori *RISC*, *Reduced Instruction Set Code*) piuttosto che su processori con un vasto repertorio di istruzioni particolari (processori *CISC*, *Complete Instruction Set Code*).

La CPU è composta da diversi elementi, tutti fisicamente assemblati sullo stesso microcircuito: l'*unità di controllo* (*control unit*) è responsabile del prelevamento delle istruzioni dalla memoria principale e della loro decodifica; l'*unità aritmetico-logica* (*Arithmetic Logic Unit* o *ALU*) è in grado di svolgere semplici operazioni aritmetiche, come somma e sottrazione di interi, e booleane, come AND e OR; questa poi è spesso coadiuvata da una *shift unit* per le operazioni di rotazione dei bit di una parola. La CPU contiene pure una piccola memoria ad alta velocità, usata per contenere valori temporanei ed alcune informazioni di controllo; tale memoria è suddivisa in *registri*, alcuni di uso generico, altri con uno scopo preciso. Il numero di registri, la loro capacità in bit ed il loro uso sono specifici di un microprocessore, per cui ci si limiterà ad informazioni generali valide per ogni processore.

Uno dei registri più importanti è il *program counter* (*PC*) che punta all'indirizzo di memoria in cui è contenuta l'istruzione successiva da eseguire; l'*instruction register* (*IR*) invece contiene una copia dell'istruzione corrente. Ogni istruzione in linguaggio macchina è rappresentata da un codice esadecimale, pertanto è semplicemente un numero che la control unit ha il compito di decodificare. Vi sono poi dei registri di uso generale impiegati per depositare temporaneamente dei dati, per esempio gli addendi di una somma; ve ne sono da un minimo di 2–3 nei microprocessori più semplici fino a 64 in quelli più potenti (i processori Intel della serie x86 ne hanno 4, detti *AX*, *BX*, *CX* e *DX*, a 16 bit nell'8086 e nel 286, a 32 bit dal 386 in poi). Tali registri possono essere utilizzati sia in lettura che in scrittura: quando un nuovo dato vi viene depositato, esso sovrascrive il precedente che perciò va perso; per conservarlo occorre prima copiarlo nella memoria principale.

Durante l'esecuzione di un programma la CPU effettua i seguenti passi: estrae dalla memoria l'istruzione da eseguire e la copia nell'instruction register, quindi aggiorna il program counter; poi determina il tipo di istruzione e se essa necessiti di un dato nella memoria, nel cui caso lo preleva e lo deposita in uno dei registri; esegue quindi l'istruzione, e salva se necessario il risultato; infine ricomincia prelevando l'istruzione successiva. Si supponga ad esempio di voler sommare due numeri contenuti in due locazioni di memoria: il programma sarà composto da una serie di istruzioni per guidare la CPU a trasferire il primo addendo in un registro, il secondo addendo in un altro registro, ad effettuarne la somma tramite la ALU (ma molti processori sono in grado di sommare direttamente un dato in memoria col contenuto di un registro particolare detto *accumulatore*), ed infine a salvare in memoria il risultato, che generalmente si trova nel registro che conteneva il primo operando.

Esistono poi alcuni registri, detti *registri di segmento*, in cui il processore conserva gli indirizzi di memoria. Infine devono essere menzionati due altri registri molto importanti, lo *stack pointer* (*SI*) ed il *program status word* (*PSW*, anche detto *flag register*). Il primo punta alla prima locazione libera nello *stack*; quest'ultimo, di cui si parlerà più diffusamente nel paragrafo dedicato agli interrupt, è un'area in cui la CPU salva informazioni relative a operazioni sospese per permettere l'esecuzione di altre istruzioni: quando il processore interrompe momentaneamente l'esecuzione di un programma per svolgerne un altro, lo stack viene usato per annotare quale operazione era in corso ed eventualmente il contenuto dei registri, affinché sia poi possibile riprendere l'esecuzione dal punto in cui era stata interrotta. I dati vengono "impilati" cioè posti sempre in cima allo stack (che in inglese significa appunto "pila") ed estratti nell'ordine inverso con apposite istruzioni in linguaggio macchina, cosicché in cima allo stack si trovano sempre le informazioni relative all'ultimo compito sospeso. Un programma può anche utilizzarlo per salvare temporaneamente altre informazioni, ad esempio per passare dati ad un sottoprogramma, ma è indispensabile che ogni dato introdotto sia poi estratto quando non serve più al fine di riportare lo stack nella condizione in cui si trovava prima.

Il program status word invece è un registro di sola lettura, i cui bit forniscono informazioni sullo stato del processore e sugli effetti dell'ultima operazione aritmetica svolta. Verificando lo stato di questi bit sono possibili test logici: infatti se il calcolatore fosse in grado solo di svolgere operazioni matematiche sarebbe poco più che una potente calcolatrice. Ciò che lo rende così versatile è la possibilità di "prendere decisioni", cioè di cambiare l'ordine di esecuzione delle istruzioni di un programma a seconda dell'evoluzione dei calcoli (le cosiddette *diramazioni condizionate*). Ad esempio, il programma potrebbe istruire la CPU ad eseguire certe operazioni piuttosto che altre se il contenuto di un certo registro, che potrebbe essere il risultato di un calcolo, sia positivo o negativo. Alcuni di questi bit, presenti in tutti i processori, sono il *bit di riporto* (*carry bit*), che è posto a 1 quando il risultato di una somma dà luogo ad un riporto, il *bit di zero*, che viene posto a 1 se l'ultima operazione ha dato un risultato nullo, il *bit di negativo*, posto a 1 se l'ultima operazione ha dato luogo ad un risultato negativo (ovvero se il bit più significativo di una parola è stato posto a 1, il che, come si è visto nel Capitolo precedente, segna appunto i numeri negativi), e il *bit di overflow*, che è posto a 1 quando l'ultima operazione ha provocato un riporto dal penultimo bit verso il bit più significativo (viene utilizzato nelle operazioni fra numeri con segno). Per confrontare due numeri la CPU sottrae il primo dal secondo: se il primo è maggiore il risultato è negativo e viene abilitato il bit di negativo; se sono uguali il risultato è nullo e viene posto a 1 il bit di zero; se il primo è minore del secondo il risultato della sottrazione è un numero positivo ed entrambi i bit restano a zero.

Così un programma può seguire strade diverse in base al risultato di questo confronto. Per testare il valore di questi bit esistono istruzioni in linguaggio macchina, solitamente dette BNE (*Branch if NEgative*) e BEQ (*Branch if EQual*).

Esistono poi uno o più *bit di interrupt* che servono appunto a gestire gli interrupt (si veda il relativo paragrafo). Infine nei processori più evoluti la PSW contiene alcuni bit che indicano la priorità e il modo di operare in cui si trova la CPU. La *priorità* è un numero che viene usato per impedire a richieste di importanza più bassa di interrompere l'esecuzione di compiti in quel momento più importanti, e anch'essa viene molto usata nella gestione degli interrupt. Il processore poi può trovarsi in diversi *stati* o *modi*, i più importanti dei quali, comuni a tutti quelli dotati di questa caratteristica, sono il *kernel mode* (o *supervisor mode*) e lo *user mode*: quando il processore è in kernel mode un programma può eseguire qualunque istruzione, mentre in user mode alcune istruzioni, in particolare quelle che coinvolgono direttamente operazioni di ingresso/uscita o accedono a certe zone riservate della memoria, non sono ammesse. La CPU esegue in kernel mode i programmi inerenti al sistema operativo, così da poter intervenire sulle periferiche e avere il pieno controllo del sistema, mentre lavora in user mode allorché esegue programmi utente, il che consente di evitare che questi ultimi interferiscano fra loro e con lo stesso sistema operativo.

La ALU è in grado di svolgere solo le operazioni più semplici fra interi. Per le operazioni più complesse, come divisioni o estrazioni di radice, e per manipolare numeri reali i primi processori erano dotati di segmenti di codice *ad hoc*: gli operandi erano depositati in opportuni registri e poi veniva chiamato un sottoprogramma che eseguiva il calcolo voluto. Un metodo più efficace è di avere un processore ausiliario, detto *coprocessore matematico* (o *Floating Point Unit*, *FPU*), che svolga tali calcoli: la CPU gli invia gli operandi e gli indica quale operazione svolgere, il coprocessore opera indipendentemente dal processore principale e alla fine gli restituisce il risultato. Il vantaggio deriva dal fatto che un coprocessore impiega per effettuare un calcolo da un cinquantesimo ad un centesimo del tempo necessario ad un sottoprogramma che lo scompone per renderlo eseguibile dal processore principale. La FPU è un processore come gli altri, con la sua unità di controllo e i suoi registri; però invece di essere di uso generico come la CPU, ha un insieme ridotto di istruzioni mirate ed è progettata per lo svolgimento di compiti ben precisi. Agli inizi si trattava di un microprocessore a sé fisicamente separato dalla CPU, ma grazie allo sviluppo della microelettronica viene oggi incorporato nella stessa CPU diventandone un elemento costitutivo, il che aumenta ulteriormente la velocità di calcolo poiché riduce i tempi di trasmissione dei segnali fra i due componenti; anche i costi di produzione vengono ad essere ridotti.

La CPU è il processore principale del calcolatore ma certo non è l'unico presente al suo interno: ve ne sono degli altri con funzioni specifiche che richiedono un insieme ristretto di istruzioni disegnato apposta. Questi processori di uso speciale sono ampiamente utilizzati per guidare le periferiche di ingresso e uscita, come dischi e schede di rete. Quando la CPU richiede il loro intervento, per esempio per leggere un dato dal disco, non attende che esso sia portato a termine, ma piuttosto invia la sua richiesta e poi prosegue i suoi compiti: sarà il processore secondario ad avvisare la CPU di aver completato l'operazione richiesta. Una importanza particolare riveste il *processore grafico*, il quale gestice la grafica del terminale video. Si tratta di un processore altamente specializzato, sempre più spesso dotato anche di una sua memoria autonoma separata dalla memoria principale del calcolatore per aumentare la sua velocità di lavoro; inoltre possiede un insieme di istruzioni dedicato che consentono con comandi semplici di disegnare punti, linee e forme geometriche di diversa intensità e colore, e perciò ben diverso dall'insieme di istruzioni di un processore di uso generico come la CPU. Un altro processore importante è il *clock*: si tratta di un oscillatore molto stabile che emette impulsi a frequenza ben determinata che vengono distribuiti agli altri processori. Ogni istruzione in linguaggio macchina si traduce in una serie di microoperazioni, ad esempio per trasferire un dato in un registro occorre che a ciascuno dei suoi bit sia assegnato un valore, il che a livello microelettronico si traduce nel cambiamento di stato di altrettanti flip-flop; affinché queste operazioni non si intralcino fra loro è necessario che avvengano con una cadenza ben precisa (che non può superare un valore massimo dipendente dal processore). Il clock ha appunto lo scopo di essere questo "metronomo" che scandisce le singole microoperazioni. Non va confuso con l'*orologio*, un altro microprocessore che alla stregua di un orologio digitale segna il giorno e l'ora correnti; esso è dotato di una batteria autonoma che gli consente di continuare a funzionare anche quando il calcolatore è spento.

Solitamente i calcolatori posseggono una sola CPU. Tuttavia vi sono calcolatori dotati di due o più CPU che lavorano in parallelo: la velocità di elaborazione viene incrementata eseguendo più istruzione allo stesso tempo. Esistono anche singoli processori capaci di eseguire contemporaneamente più di una istruzione; in genere ciò si ottiene duplicando ALU, FPU e registri.

2.3.2 La memoria principale

La *memoria* di un calcolatore è un'area in cui vengono conservate delle informazioni (sia le istruzioni di programma che i dati) mentre sono in corso altre operazioni. Il contenuto della memoria è costituito da informazioni temporanee e non permanenti: quando il calcolatore viene spento tutte queste informazioni vanno perdute (ad eccezione di quelle contenute in certe memorie a sola lettura di cui si discuterà in seguito). Per questo spesso ci si riferisce ad essa come *memoria volatile*. La memoria di un calcolatore perciò può essere paragonata ad una lavagna, appunto perché nulla vi viene scritto in modo permanente: il sistema può scrivere qualunque dato in qualunque parte (o quasi) della memoria e ciò che è stato scritto può essere modificato semplicemente riscrivendovi sopra. La memoria inoltre non fa differenza fra dati e istruzioni di programma, poiché anche queste vengono codificate sotto forma di numeri. La distinzione tra programma e dati, che è essenziale per il processore, non viene presa in considerazione dalla memoria: qualsiasi informazione viene salvata con una particolare disposizione di bit, e la stessa disposizione di bit può indicare un dato numerico, una lettera o una istruzione in linguaggio macchina. Così gli stessi byte di memoria vengono utilizzati per registrare programmi e dati numerici e alfanumerici.

La memoria consiste di un certo numero di *celle* o *locazioni*, ognuna delle quali può contenere una parte di informazione. Ogni locazione di memoria contiene lo stesso numero di bit. Tale numero può essere qualsiasi (teoricamente sono possibili anche memorie con celle da 1 solo bit), ma di solito le celle contengono 8 o 16 bit, cioè 1 o 2 byte. Ciascuna cella ha un suo *indirizzo* che la identifica in modo univoco; celle adiacenti hanno indirizzi consecutivi. Anche gli indirizzi sono espressi mediante numeri binari: se un indirizzo ha m bit il numero massimo di locazioni indirizzabili direttamente è 2^m, ed è indipendente dal numero di bit per cella. Gli stessi indirizzi poi essendo dei numeri possono essere a loro volta contenuti in altre locazioni di memoria.

Le istruzioni di programma vengono ovviamente memorizzate nel loro ordine "naturale" cioè nell'ordine in cui vanno eseguite. Lo stesso vale per insiemi generici di byte come ad esempio le stringhe di caratteri. Quando invece si tratta di numeri, interi o reali, a 2 o a 4 byte, sistemi diversi si comportano in modo diverso, talora opposto all'ordine intuitivo. Si consideri per esempio il numero 16F0 (5872 in notazione decimale), che risulta composto dai due byte 16 e F0. Alcuni sistemi scrivono in memoria prima il byte più significativo e poi il meno significativo, cioè lo salvano come 16 F0: questo formato dei dati è detto *big endian data format*. Altri sistemi invece scrivono prima il byte meno significativo e poi quello più significativo, cioè lo salvano come F0 16: questo formato è detto *little endian data format**. Pertanto ogniqualvolta si manipolano direttamente i dati in memoria occorre stare attenti al formato usato dal calcolatore.

La CPU comunica con la memoria tramite il bus (si veda il relativo paragrafo). Un accesso alla memoria è in genere molto più lungo del tempo richiesto al processore per eseguire una microoperazione, quindi dati e indirizzi devono essere mantenuti disponibili al bus per un certo tempo. Per facilitare questo compito la CPU non comunica con la memoria direttamente ma attraverso due registri ausiliari, posti a cavallo fra la CPU e il bus, il *Memory Address Register* (*MAR*), che contiene gli indirizzi, ed il *Memory Buffer Register* (*MBR*), che contiene i dati. Quando la CPU necessita di leggere una cella, sia che contenga un'istruzione o un dato, ne carica l'indirizzo nel MAR e invia una richiesta di lettura alla memoria tramite il bus; dopo un ciclo di memoria questa pone il contenuto della cella nel MBR, da cui la CPU lo può prelevare. Viceversa per scrivere in una cella la CPU ne carica l'indirizzo nel MAR e il dato da registrare nel MBR, quindi invia una richiesta di scrittura, che viene eseguita nel ciclo di memoria successivo. In alcuni calcolatori la memoria lavora in modo sincrono, cioè deve completare l'operazione di lettura o scrittura in un tempo prefissato; in altri invece la memoria può prendersi il

* Pare che l'origine di queste espressioni risalga al romanzo "I viaggi di Gulliver" di Jonathan Swift: in esso il protagonista incontra una popolazione divisa in due fazioni in guerra fra loro; una di esse sostiene che le uova vadano rotte dalla loro estremità più grande (*big end*) e perciò sono detti *big–endians*, i secondi invece che vadano rotte dall'estremità più piccola (*little end*) e per questo detti *little–endians*.

tempo che vuole, limitandosi poi a segnalare, ancora tramite il bus, il completamento della richiesta.

Come si è detto in precedenza, se gli indirizzi sono numeri a m bit si possono indirizzare direttamente al più 2^m locazioni di memoria (per esempio, con numeri a 16 bit si indirizzano al massimo $65536 = 64$K celle). Il processore tuttavia può gestire un numero maggiore di celle utilizzando una tecnica chiamata *indirizzamento segmentato*: l'indirizzo effettivo di una locazione, detto *indirizzo assoluto*, viene scomposto in due parti, dette *indirizzo di segmento* e *scarto* (o *offset*), in modo tale che entrambi siano numeri a m bit. Il primo indica il gruppo di celle, detto *segmento*, in cui si trova la cella voluta, mentre il secondo ne fornisce la posizione esatta all'interno di un segmento. In tal modo si può arrivare fino a 2^{2m} indirizzi ($4.29 \cdot 10^9 = 4$G celle con 16 bit). Questi indirizzi sono conservati in appositi registri del processore, i registri di segmento.

Le memorie di cui si è trattato finora, le cui celle possono essere sia scritte che lette, vengono genericamente indicate col termine *RAM*, per *Random Access Memories*, cioè memorie ad accesso casuale (il nome è inesatto giacché anche agli altri tipi di memoria si può accedere in modo casuale, ovvero ogni cella può essere raggiunta indipendentemente dalle altre). Le RAM possono essere *statiche* o *dinamiche*: le prime sono costruite usando circuiti che possono mantenere l'informazione per un tempo indefinito, purché venga loro fornita l'alimentazione; le seconde invece sono costituite da schiere di piccoli condensatori, e poiché questi tendono a scaricarsi entro alcuni millisecondi devono essere periodicamente "rinfrescate". Le memorie dinamiche sono perciò più complesse, anche se di solito hanno una capacità maggiore.

Vi sono però delle informazioni che devono essere mantenute permanentemente in memoria anche quando il calcolatore viene spento. Si tratta di dati e programmi che una volta installati non necessitano più di essere modificati. Per queste funzioni vengono adoperate memorie *ROM* (*Read–Only Memories* cioè memorie a sola lettura), le quali non possono essere alterate o cancellate né intenzionalmente né accidentalmente: l'unico modo per rimpiazzare i dati di una ROM è sostituire l'intero chip elettronico. Le memorie ROM di un calcolatore vengono utilizzate per registrarvi un insieme di programmi di utilità, detto *BIOS*. Esso sarà trattato in modo più approfondito nel prossimo Capitolo, per ora basti dire che una parte di questi programmi viene eseguita all'accensione del calcolatore, mentre altri facilitano la gestione del sistema e delle sue periferiche.

I dati in una memoria ROM vengono inseriti durante la sua fabbricazione, essenzialmente esponendo un materiale fotosensibile alla luce attraverso una maschera contenente lo schema di bit desiderato e poi asportando con reagenti chimici la parte non esposta. Le ROM sono in genere più economiche delle RAM, ma non sono flessibili non potendo più essere alterate. Per ovviare in parte a questo problema sono state sviluppate le cosiddette *PROM* (*Programmable ROM*), che possono essere programmate una sola volta come una RAM ma poi restano permanenti come le ROM. Una ulteriore evoluzione è rappresentata dalle *EPROM* (*Erasable PROM*), le quali possono essere cancellate e riprogrammate più volte; la cancellazione però è una operazione complessa, poiché richiede l'esposizione del chip, attraverso una finestra di quarzo di cui è dotato, ad una intensa luce ultravioletta per diversi minuti. Ancora migliori sono le memorie *EEPROM* (*Electrically Erasable PROM*), dette anche *EAROM* (*Electrically Alterable ROM*), il cui contenuto può essere cancellato applicando opportuni impulsi elettrici. Quindi una EEPROM può essere riprogrammata come una RAM, ma il suo contenuto non è volatile; tuttavia il tempo richiesto da una EEPROM per scrivere o cancellare un byte è migliaia di volte più lungo di quello impiegato da una RAM, il cui vantaggio perciò è di essere enormemente più veloce in scrittura. Il tempo di accesso in lettura è invece praticamente lo stesso per le memorie RAM come per i varî tipi di ROM (dell'ordine di alcune centinaia di nanosecondi). Le EEPROM hanno molti ambiti di applicazione al di fuori dei calcolatori, ogniqualvolta è necessario alterare delle informazioni che poi si vogliono non volatili (si pensi ad esempio ai settaggi dei canali di un televisore).

2.3.3 Le memorie di massa

Le memorie ad accesso casuale sono costose e hanno una capacità relativamente limitata. Per immagazzinare in modo permanente grandi quantità di dati si usano le *memorie di massa*; queste in genere (ma non sempre) usano tecniche di registrazione magnetica ed hanno appunto lo scopo di archiviare dati e di permetterne il recupero anche a distanza di tempo. Il rovescio della medaglia è una velocità di accesso decisamente inferiore.

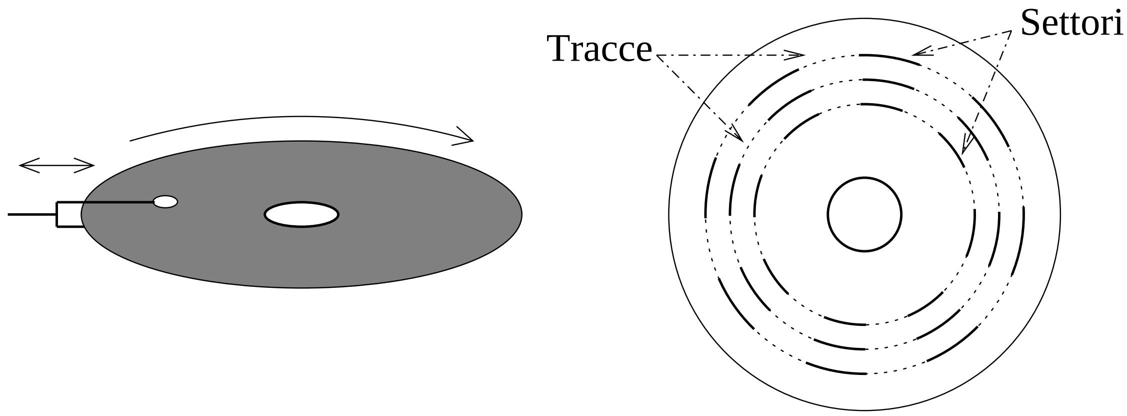

Fig. 2.3 *Elementi essenziali di un disco magnetico*

La registrazione magnetica utilizza lo stesso metodo usato in tutti i supporti magnetici, come le cassette audio e video, e si basa sulla possibilità che composti del Ferro e altri materiali possano essere magnetizzati. Questa tecnica apparve all'inizio per la registrazione dei suoni in forma *analogica*, e successivamente fu sviluppata e adattata alla registrazione di informazioni in forma *digitale*, che è quella richiesta da un calcolatore.

La registrazione avviene su una superficie sensibile, di solito composta da ossido di Ferro che le conferisce il tipico colore marrone ruggine. Tale superficie ricopre un materiale di supporto, generalmente mylar (usato per nastri e dischetti) o alluminio rigido (per i dischi fissi); più il rivestimento magnetico è sottile e meglio funziona.

Qualunque sia il dispositivo di registrazione, le informazioni vengono memorizzate secondo lo stesso schema: la superficie sensibile viene considerata come un insieme di punti, ciascuno dei quali rappresenta un bit e viene posto a 0 o a 1 (che corrispondono a due diversi stati di magnetizzazione). Lo schema di registrazione inoltre si serve di alcuni *segni di sincronizzazione* per localizzare l'esatta posizione dei punti: questo spiega perché qualsiasi unità di questo tipo debba prima essere "preparata" a contenere i dati (o, come si suol dire, *formattata*) con una procedura più o meno complessa. Il grande vantaggio della registrazione magnetica è la possibilità di cambiare in modo relativamente facile il contenuto delle informazioni archiviate.

Fra i sistemi di memorizzazione di massa i dischi magnetici sono senz'altro i più importanti e diffusi. Tutti i calcolatori moderni contengono almeno un *disco fisso* o *disco rigido* (*hard disk*). Si tratta essenzialmente di un disco di materiale rigido ricoperto su entrambi i lati da una sostanza magnetizzabile; esso ruota a velocità costante in mezzo a due testine di registrazione, ognuna delle quali è in grado di cambiare lo stato di magnetizzazione della porzione di disco sulla quale si trova. Le testine a loro volta possono muoversi radialmente: il moto combinato del disco e delle testine permette di raggiungere qualsiasi porzione del disco stesso (Fig. 2.3). I dischi infatti sono unità *ad accesso diretto*, in quanto è possibile esaminare le informazioni contenute in un punto qualsiasi indipendentemente dalla sua posizione.

In pratica però ciascuna unità è costituita non da un singolo disco bensì da più dischi impilati che ruotano sullo stesso asse; un braccio meccanico sostiene tante coppie di testine quanti sono i dischi, ciascuno dei quali ruota in mezzo ad una coppia. Il tutto è chiuso all'interno di un contenitore sigillato che li protegge dalla polvere e che funge da supporto per i circuiti di alimentazione e di gestione dei movimenti dei dischi e delle testine; tali circuiti permettono anche lo scambio dei dati col calcolatore. Ogni disco avrà quindi un *numero di lati* (o *head number*, cioè numero di testine) caratteristico; i lati sono numerati a partire da zero.

In un disco magnetico le informazioni vengono registrate su cerchi concentrici, detti *tracce* (*tracks*): la superficie dei dischi è suddivisa in tracce distinte dall'esterno verso l'interno (Fig. 2.3); anch'esse vengono numerate da zero, che è la traccia più esterna. Il numero di tracce nei dischi fissi è molto variabile, ma comunque non sono contenute su tutta la superficie registrabile ma solo su una parte; la densità di tracce viene espressa in *tracce per pollice* (*tracks per inch* o *TPI*). Le tracce sono a loro volta suddivise in *settori* (*sectors*); il numero di settori per traccia varia a seconda del tipo e del formato del disco, ma tutti i settori di uno stesso disco hanno la stessa capacità in byte. Il numero di byte per settore può essere qualunque, ma 512 byte/settore sono diventati di fatto la dimensione standard. I settori di una traccia sono sì numerati anch'essi a partire da zero, ma il settore 0 di ciascuna traccia è riservato per funzioni di identificazione e non può contenere dati, per cui di fatto è come se fossero numerati da 1 in avanti. Tutte le operazioni di lettura e scrittura vengono effettuate dal calcolatore in termini di settori completi: sebbene, come è ovvio, sul disco possono essere salvati dati di qualsiasi dimensione, le operazioni di input/output di basso livello sono eseguite solo per settori completi.

Numero di lato, numero di traccia e numero di settore nella traccia identificano univocamente ogni settore del disco, che può essere raggiunto mediante rotazione dei dischi e movimento del braccio porta–testine (in realtà i dischi sono sempre in rotazione, e il sistema aspetta che il settore voluto passi al di sotto delle testine). Spesso però si fa riferimento ad un'altra quantità, il *cilindro* (*cylinder*): un cilindro è l'insieme di tutte le tracce, una per lato, che si trovano alla stessa distanza radiale dal centro del disco. Quindi un settore può essere identificato da numero di lato, numero di cilindro e numero di settore. Poiché il tempo necessario a spostare il braccio porta–testine fra due tracce (il cosiddetto *seek time*) è mediamente molto maggiore del tempo necessario affinché due settori diversi della stessa traccia passino al di sotto delle testine (*rotational latency*), il sistema tende a registrare i raggruppamenti omogenei di dati su settori, possibilmente contigui, di una stessa traccia: ovvero, tendenzialmente i dati vengono scritti su settori di uno stesso cilindro, in quanto le operazioni di scrittura e lettura possono avvenire senza muovere le testine*.

Moltiplicando il numero di lati per il numero di tracce per lato e per il numero di settori per traccia si ottiene il numero totale di settori; moltiplicando quest'ultimo per il numero di byte per settore si ricava la capacità di stoccaggio di un disco. I primi dischi rigidi contenevano pochi MB di dati, oggi si arriva facilmente alle decine di GB; tuttavia occorre tener presente che una piccola frazione di questo spazio viene usata dal sistema operativo per la gestione del disco stesso. Quando però si presenta la necessità di avere capacità ancora maggiori, i calcolatori di medie e grandi dimensioni possono far uso di un *RAID* (*Redundant Array of Inexpensive Drives*), che consiste in un gruppo di dischi fissi che funzionano insieme come se fossero un disco unico. Un RAID tipicamente contiene cinque o più dischi, che assicurano una velocità di accesso ragionevolmente alta e sono più affidabili (e talora meno costosi) di un tradizionale disco singolo di pari capacità.

Come si è detto all'inizio, un supporto magnetico non può essere utilizzato se prima non viene formattato. Nel caso dei dischi in realtà si hanno due livelli di formattazione,

* Questa possibilità non viene sfruttata da tutti i sistemi operativi: il DOS ad esempio non ne fa uso, perché internamente numera tutti i settori in modo sequenziale ignorando la struttura "tridimensionale" costituita da lato/cilindro/settore.

la formattazione fisica e la formattazione logica. La *formattazione fisica* consiste nella creazione delle tracce e dei settori completi degli indirizzi necessarî per identificarli; la porzione di settore utilizzabile dall'utente viene invece riempita di dati insignificanti. Generalmente un disco fisso viene venduto già formattato fisicamente. Con la *formattazione logica* invece il disco viene preparato per essere gestito da un determinato sistema operativo, mediante la creazione di una struttura logica che gli consenta di localizzarne le varie parti. Questo processo serve inoltre a identificare i blocchi difettosi, cioè imperfezioni del supporto che ne rendono alcune zone non affidabili per registrarvi dati, le quali vengono quindi marcate per non essere utilizzate.

I dischi fissi portano questo nome giacché una volta installati in un calcolatore rimangono permanentemente al suo interno. Esistono tuttavia anche dischi rigidi *estraibili*, che possono essere facilmente estratti e inseriti in altri calcolatori oppure conservati in archivio.

Decisamente più diffusi sono i cosiddetti *dischetti* o *floppy disk*: si tratta di un unico disco magnetico di materiale plastico flessibile (da cui il nome inglese) custodito in un contenitore rigido quadrato di 3.5" di lato; questo involucro al suo interno è rivestito da un materiale speciale che aiuta il disco a ruotare dolcemente e garantisce la pulizia della sua superficie. Per poterlo utilizzare va inserito in un *lettore* opportunamente dotato di una testina che lo legge e scrive attraverso una scanalatura praticata nella custodia; esso inoltre provvede alla rotazione del disco. Anche un dischetto deve essere formattato: dopo la formattazione fisica esso contiene 80 tracce per una capacità massima di 1.44 MB. Quasi completamente caduto in disuso invece è un altro tipo di dischetto, di 5.25" di lato e 1.2 MB di capacità massima; di recente poi è stato sviluppato un nuovo tipo di dischetto e relativo lettore, detto *ZIP*, avente esternamente le stesse dimensioni dei floppy disk da 3.5", ma che può contenere fino a 120 MB. In tutti i casi comunque i concetti di base circa la loro struttura sono gli stessi dei dischi rigidi.

Ogni unità di questo tipo, sia essa disco fisso o lettore di dischetto, è dotata di un processore dedicato, il *disk controller*, che sovraintende al trasferimento di dati tra la memoria principale e i dischi stessi. Per specificare un trasferimento, il programma in esecuzione deve fornire al controller alcune informazioni: il settore in cui sono contenuti i dati voluti, il numero di parole da trasmettere e l'indirizzo di memoria in cui o da cui vanno trasferiti; quindi il processore invia tramite il bus la richiesta di lettura o scrittura al controller. Ed è lo stesso controller ad avvisare il processore, ancora tramite il bus, di aver effettuato il trasferimento.

Un altro supporto magnetico molto diffuso sono i *nastri magnetici*. Essenzialmente si tratta della stessa tecnica utilizzata per la registrazione di suoni: un nastro di materiale plastico rivestito di una sostanza magnetizzabile scorre al di sotto di una testina di registrazione. La differenza sostanziale è che le informazioni sono scritte in forma digitale e non analogica. Un tempo sui nastri veniva scritto un byte alla volta: quindi esistevano 9 piste, 8 per gli altrettanti bit di un byte più una per il *bit di parità*, un bit supplettivo che aumenta l'affidabilità dei dati in fase di lettura. Al giorno d'oggi i nastri portano 16 o 32 tracce (più il bit di parità) cioè registrano 2 o 4 byte alla volta. La densità di dati sul nastro viene perciò misurata in *byte per pollice* (*byte per inch* o *bpi*). Dati omogenei vengono riuniti in gruppi detti *record*; fra un record e il successivo l'unità lascia uno spazio vuoto non registrato, il che facilita l'individuazione di questi gruppi durante la lettura. Pure le unità nastro sono dotate di controller, ancorché la struttura dei dati e quindi le procedure di trasferimento siano più semplici.

Diversamente dai dischi i nastri magnetici sono memorie di massa *ad accesso sequenziale*, cioè per raggiungere un certo gruppo di dati è necessario passare in rassegna tutti quelli che li precedono: ad esempio, per leggere il record n l'unità nastro deve prima leggere tutti i record da 1 a $n - 1$. Sono quindi più adatti a contenere dati che per loro natura vanno riletti sequenzialmente, oppure che una volta salvati andranno riletti solo in caso di necessità, come per esempio le copie di sicurezza dei dati su disco.

Tutti i supporti magnetici sono molto sensibili ai campi elettrici e magnetici esterni, che possono danneggiarli. Anche la semplice esposizione alla radiazione naturale degrada col tempo la qualità dei dati registrati, sino a renderli dopo alcuni anni illeggibili: ad esempio, i nastri di buona qualità generalmente durano per cinque o più anni, ma un normale dischetto non dovrebbe essere utilizzato per conservare dati per più di un paio.

Al contrario i *dischi ottici* sono in grado di mantenere intatta l'informazione per un periodo decisamente maggiore. I *CD-ROM* (*Compact Disc Read–Only Memory*) sono di certo i più diffusi; in essi i dati sono immagazzinati come una serie di microscopiche cavità ottenute per stampaggio su un disco di plastica. Per la lettura viene utilizzato un laser ottico di bassa potenza: una piccola fotocellula rileva le variazioni di intensità della luce riflessa dalle microcavità e le converte in segnali elettrici.

I supporti ottici raggiungono una capacità di memoria per unità di area indubbiamente maggiore dei supporti magnetici, poiché un fascio laser può essere controllato e focalizzato con molta maggior precisione che una seppur minuscola testina magnetica, permettendo di condensare i dati in uno spazio più piccolo. Inoltre sono poco costosi da realizzare trattandosi semplicemente di dischi di plastica pressati da uno stampo. I dati registrati su di essi non vengono alterati da disturbi elettromagnetici, e i dischi stessi sono più resistenti ai danni fisici di quanto non lo siano dischi e nastri magnetici, che vanno conservati in contenitori sigillati al riparo da polveri e contaminanti.

Il principale difetto dei CD-ROM è che sono supporti a sola lettura (da cui il nome): per questo vengono usati per distribuire informazioni "statiche" come enciclopedie, database, programmi e altro software per calcolatore. Nei primi anni '90 fu introdotto un nuovo tipo di CD-ROM, il *CD-Recordable* o *CD-R*: esso è dotato al suo interno di una strato costituito da un polimero sensibile alla luce, che può essere letteralmente "bruciato" da un raggio laser dando luogo a piccole macchie scure le quali facendo le veci delle microcavità possono essere rilette da normali lettori di CD-ROM. Potendo essere scritti una sola volta ma riletti innumerevoli volte (per questo vengono anche detti *WORM* cioè *Write Once Read Many*), questi dischi costituiscono una valida alternativa ai supporti magnetici per il salvataggio di quei dati che si vorranno rileggere più volte nel tempo, pur non garantendo la stessa durata degli ordinari CD-ROM. Un ulteriore miglioramento è stato ottenuto a metà degli anni '90 sostituendo il polimero fotosensibile con dei cristalli fotoeccitabili: il supporto così ottenuto, noto come *CD-RW*, può essere cancellato e riscritto più volte utilizzando sempre luce laser.

A metà strada fra i due tipi di supporto si trovano i *dischi magneto–ottici*, nei quali si sfrutta l'interazione fra luce e campi magnetici. Le informazioni vi vengono scritte mediante un intenso fascio laser che è in grado di cambiare la direzione di magnetizzazione dei punti della superficie del disco che rappresentano i singoli bit. In fase di lettura la luce prodotta da un laser di bassa potenza viene riflessa con diversa polarizzazione da punti aventi diversa direzione di magnetizzazione.

Altre memorie di massa sono invece cadute in completo disuso; se ne ricorderanno alcune per ragioni storiche, poiché hanno avuto la loro importanza agli inizi dello sviluppo dei calcolatori elettronici. I *tamburi magnetici* sono una di queste: si trattava di un cilindro ruotante attorno al proprio asse e la cui superficie laterale era rivestita di una sostanza magnetizzabile; una o più testine di registrazione erano usate per la lettura e la scrittura. Il principio di funzionamento è lo stesso dei dischi magnetici: anch'essi erano unità ad accesso diretto leggibili e scrivibili a piacere. Le *schede perforate* sono state usate a lungo per introdurre dati e programmi in un calcolatore; si trattava di schede di cartoncino sui quali i dati erano rappresentati da opportune combinazioni di fori. I *nastri perforati* invece erano lunghi rotoli di carta su cui i fori, disposti in verticale, rappresentavano i bit di un byte. Schede e nastri perforati erano supporti ad accesso sequenziale, e ovviamente le informazioni una volta registrate non potevano più essere modificate.

Come si è detto, i dischi comunicano col processore e con la memoria tramite il bus. Esistono diversi standard per gestire le comunicazioni fra dischi e bus, ma quelli di gran lunga più diffusi sono lo standard IDE/EIDE, usato quasi esclusivamente sui personal computer, e lo standard SCSI, usato soprattutto per i calcolatori di dimensioni maggiori ma talora anche sui personal.

Il termine *Integrated Device Electronics* (*IDE*) fu coniato nella seconda metà degli anni '80 per designare un nuovo disk controller integrato su un singolo chip introdotto dalla IBM. Questo standard si diffuse rapidamente ed è tuttora adottato seppure con alcuni miglioramenti: nei primi anni '90 infatti fu sviluppato l'*Enhanced IDE* (*EIDE*) il cui progresso principale consiste nell'aver portato la velocità massima di trasferimento dei dati da 1–2 Mb/s a 12 Mb/s.

L'interfaccia IDE/EIDE è più semplice e meno costosa. Le unità sono connesse al disk controller mediante degli *adattatori*; ogni adattatore collega due dischi, dei quali uno solo può essere il *principale* (*master*) mentre l'altro è il *secondario* (*slave*); la scelta fra le due possibilità avviene mediante il posizionamento di microinterruttori o ponticelli di cui le unità sono dotate. Uno solo dei due può essere occupato ad un certo istante: questo rappresenta una grave limitazione dello standard IDE/EIDE, perché se il processore ha necessità di accedere ad entrambi i dischi non può farlo in parallelo ma deve attendere che uno dei due abbia completato il trasferimento per poter accedere all'altro (solitamente l'unità principale ha la precedenza), il che aumenta i tempi morti. Un controller IDE ha spazio per due adattatori e quindi può gestire al massimo quattro dischi; un controller EIDE arriva a quattro adattatori. Inoltre per collegare periferiche diverse dai dischi, come stampanti, scanner o unità nastro, sono necessari adattatori particolari diversi per ogni unità.

L'interfaccia *SCSI* (*Small Computer System Interface*) è più costosa e necessita di un controller dedicato, ma permette di raggiungere prestazioni decisamente superiori. Lo standard SCSI si limita a definire un protocollo generico per il trasferimento dei dati senza alcuna informazione specifica circa i dispositivi che lo utilizzeranno: ciò permette di far uso dello stesso collegamento per periferiche molto diverse fra loro come dischi, nastri, stampanti, scanner e lettori di vario tipo.

Le unità SCSI sono collegate in cascata una dopo l'altra (una configurazione nota come *daisy–chain*); ogni unità generalmente è dotata di due prese SCSI, ognuna delle quali può essere indifferentemente l'ingresso o l'uscita. Questa catena deve essere "terminata" ad entrambi gli estremi: di solito l'estremo interno al calcolatore è automaticamente terminato dal controller, mentre per l'estremo esterno si usa una spina particolare, il *terminatore* (*chain terminator*), da inserire in una delle due prese (alcune periferiche possiedono una sola presa SCSI avendo già il terminatore al loro interno; queste unità vanno ovviamente poste al fondo della catena). In una catena possono essere collegate al massimo otto unità compreso il controller, vale a dire che un controller SCSI può gestire fino a sette periferiche SCSI. Ognuna di esse ha assegnato un numero, l'*indirizzo SCSI* (*SCSI address* o anche *target number*), che la identifica in modo univoco. Gli indirizzi vanno da 0 a 7; abitualmente, ma non necessariamente, il controller ha l'indirizzo 7. Un indirizzo SCSI è essenzialmente arbitrario: tecnicamente esso determina la priorità della periferica sulla catena, ma in pratica la priorità esatta non fa molta differenza; importante è solo che due unità non abbiano lo stesso indirizzo, altrimenti come si può immaginare si crea un conflitto e nessuna delle due può essere comandata. In genere un dispositivo SCSI è dotato di un selettore per assegnargli l'indirizzo.

Oltre a consentire un facile collegamento di periferiche di tipo diverso, lo standard SCSI permette di raggiungere maggiori velocità di trasferimento dei dati, anche grazie a tecniche avanzate come il *Bus Master*, che rappresenta la possibilità per il controller SCSI di assumere il controllo del bus e iniziare un trasferimento senza carico per la CPU, che rimane così libera di eseguire altre operazioni. Inoltre questo standard ammette l'accesso in parallelo alle periferiche: il processore dopo aver inviato un comando ad una di esse può dialogare con un'altra senza attendere che la prima abbia portato a termine la richiesta.

2.3.4 I dispositivi di ingresso e uscita. Le porte

Il microprocessore e la memoria di per sé costituiscono già un sistema funzionante; ma sarebbe un sistema inutile se non potesse comunicare con l'ambiente esterno, dal quale vengono immessi nel calcolatore i dati iniziali e al quale esso deve fornire i risultati. A questa comunicazione provvedono i *dispositivi di ingresso e uscita* (o *di input/output*, *I/O*): si tratta di una vasta classe di periferiche che consentono gli scambi di dati non solo fra calcolatore e operatore umano, ma anche con altri calcolatori o con strumenti di misura e apparati varî (a solo titolo di esempio, si pensi ad un impianto industriale controllato da calcolatore: questo deve poter ricevere informazioni da sensori e apparecchi di misura ed inviare comandi a macchinari di produzione e di controllo; lo stesso dicasi per la raccolta dati e la supervisione di un apparato sperimentale). Alcuni Autori considerano anche i dischi e i nastri fra le periferiche di I/O: in effetti tutti i supporti estraibili possono essere utilizzati come mezzi per scambiare informazioni fra calcolatori, per inserire dati e programmi e per salvarne i risultati; tuttavia appare più logica la loro sistemazione fra i dispositivi per lo stoccaggio permanente di informazioni (questo esempio comunque deve mettere in guardia contro le suddivisioni e gli schemi rigidi, utili da un punto di vista didattico ma che spesso sacrificano alcuni aspetti per sottolinearne altri).

Vi sono innumerevoli dispositivi di comunicazione con l'esterno, fra cui stampanti, scanner, lettori ottici di testi, microfoni e videocamere solo per citare i più diffusi, nonché modem e schede di rete, di cui si tratterà in un capitolo a parte, ma quelli di gran lunga più utilizzati sono la tastiera e lo schermo video. La tastiera è un dispositivo unidirezionale poiché con essa è possibile solo inserire dati nel calcolatore. Ciò avviene attraverso un controller dedicato installato direttamente sulla piastra madre; esso è più che una semplice interfaccia e spesso controlla anche i dispositivi di puntamento come il mouse.

Quando viene premuto un tasto, la tastiera segnala l'evento al processore mediante la generazione di un interrupt (si veda il paragrafo relativo); al tempo stesso salva il *numero di identificazione* (o *scan code*) del tasto in una memoria dedicata detta *buffer di tastiera* (*keyboard buffer*), in cui può essere annotata la pressione di più tasti se la CPU tarda a rispondere all'interrupt (che solitamente ha una bassa priorità). Va notato che la tastiera rileva non solo la pressione di un tasto ma pure il suo rilascio: entrambe queste azioni vengono memorizzate e generano un interrupt.

Al segnale di interrupt la CPU sospende il programma in corso e inizia l'esecuzione di una procedura di gestione contenuta nel BIOS. Tale procedura richiede alla tastiera il codice dei tasti premuti, e questa glieli comunica azzerando poi il buffer. La procedura di gestione innanzitutto verifica se il codice corrisponde ad uno dei tasti di controllo, come *Shift*, *Alt* o *Control*, e in caso positivo ne tiene nota giacché la loro pressione o il loro rilascio serve a modificare il significato dei tasti successivamente premuti (per esempio se *Shift* è premuto le lettere vanno interpretate come maiuscole). Poi controlla se sono stati premuti alcuni tasti speciali, come *Pausa* (*Pause*), che sospende l'esecuzione del programma in corso, o *Stampa* (*Print Screen*), che stampa la schermata video attuale, oppure alcune combinazioni di tasti, come *Ctrl–Alt–Del* che nei personal computer determina il riavvio del sistema: in questi casi viene generato un secondo interrupt la cui procedura di gestione svolge il compito particolare richiesto. Se tutti questi test hanno esito negativo, il tasto premuto va interpretato come semplice carattere. La traduzione da scan code a carattere è però una operazione software svolta dal sistema operativo e non dalla tastiera, e ciò consente di cambiare con facilità il significato dei tasti. Vi sono diversi standard di conversione da scan code a carattere, che spesso tengono conto anche delle particolarità dei singoli alfabeti (come segni diacritici, lettere particolari e così via), ma quello di gran lunga più diffuso è il *codice ASCII* descritto in Appendice.

Contrariamente alla tastiera il video può essere un dispositivo di comunicazione bidirezionale: normalmente infatti viene usato solo per presentare i dati in uscita, ma con opportuni dispositivi come le *penne ottiche* è possibile anche utilizzarlo per fornire dati in ingresso; esistono poi schermi particolari, detti *touch screen*, con i quali l'informazione viene inserita semplicemente toccandone la superficie.

Per aumentare al massimo la velocità e la flessibilità del calcolatore i dati da visualizzare sullo schermo sono conservati in una memoria dedicata inserita al suo interno. Gli indirizzi di questa memoria fanno parte dello spazio di indirizzi a disposizione del calcolatore, il che ne facilita l'accesso da parte del sistema operativo. La memoria video ha però una particolarità che la distingue dal resto della memoria RAM e ROM: mentre questa comunica solo col processore, alla memoria video deve poter accedere anche la scheda che gestisce lo schermo allo scopo di visualizzarne il contenuto. Un circuito dedicato consente alla CPU di scrivervi i dati da presentare e alla scheda video di leggerli senza che i due componenti si intralcino fra loro, e questo grazie all'uso di due porte diverse per le comunicazioni (per ciò questo tipo di memoria viene detta *dual port memory* cioè *memoria a due porte*).

Il video di un calcolatore funziona in maniera simile ad un televisore: lo schermo viene costantemente "disegnato" da un fascio di elettroni in movimento; mentre il fascio scorre lungo lo schermo il controller video che lo guida continua a leggere i dati dalla memoria video e li converte in segnali che controllano il fascio di elettroni.

Il calcolatore può trovarsi in diverse *modalità video* che gli permettono di visualizzare i dati in modi differenti. Ogni controller ne possiede più d'una e possono essere cambiate con comandi software. Esse possono essere suddivise in due grandi categorie, le *modalità testo* e le *modalità grafiche*. In modalità testo è possibile visualizzare solo i caratteri del codice ASCII; essi poi non possono essere posizionati in un punto qualsiasi dello schermo: questo infatti risulta suddiviso in una griglia di 80 colonne per 25 righe ed ogni carattere può trovare posto solo in una delle 2000 locazioni risultanti. Il sistema operativo può controllare solamente la posizione all'interno della griglia e il codice del carattere ma non ne può cambiare l'aspetto, le dimensioni o il formato né come si è detto farlo apparire in una posizione intermedia. In modalità grafica invece lo schermo è suddiviso un una griglia molto più fitta di punti elementari detti *pixel* (da *picture elements*, *elementi grafici*), ognuno dei quali può essere singolarmente controllato. Poiché qualsiasi cosa appaia sullo schermo è il risultato di una combinazione di questi elementi, in modalità grafica è possibile rappresentare non solo caratteri di aspetto, forma e dimensioni diversi, ma disegni, forme geometriche e grafici qualsiasi, ognuno dei quali può trovarsi in una posizione arbitraria, e questo perché il sistema è in grado di controllare tutte le caratteristiche (come colore e intensità) di ciascun pixel. Le varie modalità grafiche si distinguono fra loro per la quantità di punti a disposizione, quantità che prende il nome di *risoluzione*: tutte le schede video moderne ne hanno più di una, da un minimo di 640 punti in orizzontale per 480 in verticale a 1280 per 1024 e oltre. Anche il numero totale di colori a disposizione può essere variato da 16 a molti milioni.

Per dialogare con l'esterno il calcolatore ha poi a disposizione anche le porte seriale e parallela. Occorre far notare che il termine “porta” ha in realtà due significati: con esso infatti si intende sia il dispositivo fisico tramite il quale avviene il collegamento tra la piastra madre ed una periferica, sia il meccanismo logico con cui il microprocessore interagisce con un dispositivo di I/O; in quest'ultimo caso le porte sono contrassegnate da un numero che indica a quale dispositivo sono associate.

La *porta parallela* è così detta perché consente di trasmettere otto bit alla volta. Questo tipo di porta viene usato quasi esclusivamente per il collegamento di stampanti. Poiché però le porte parallele possono sia ricevere che trasmettere dati, è possibile utilizzarle anche per altri tipi di comunicazione: ad esempio esistono lettori di dischetti ZIP che vengono connessi al calcolatore mediante la porta parallela. L'interfaccia usata da questa porta non è molto sofisticata e permette lo scambio di pochi segnali di controllo; altri segnali però possono essere codificati nei dati trasmessi. Va comunque notato che le caratteristiche e le prestazioni delle interfacce parallele utilizzate dalle stampanti sono state definite in modo da poter essere utilizzate da svariati strumenti elettronici e non solo dai calcolatori.

Analogamente la *porta seriale* trasmette i bit in serie, uno alla volta. Lo standard su cui si basa si chiama RS-232 e consente la trasmissione dei dati in ambo i sensi in modo asincrono (asincrono significa che la trasmissione o la ricezione non seguono una scansione predefinita; in una trasmissione sincrona invece i dati si susseguono ad intervalli regolari). A differenza della porta parallela, che è stata concepita principalmente per trasmettere dati ad una stampante, l'interfaccia seriale permette una grande quantità di impieghi diversi. Essa pertanto è molto più flessibile, ma proprio per questo il suo uso richiede la precisa definizione di un gran numero di *parametri di comunicazione*, il che a volte ne rappresenta una complicazione. Questi parametri costituiscono le specifiche che definiscono il funzionamento di ciascuna delle porte tra cui viene realizzato il collegamento, fornendo ad esse le istruzioni necessarie al riconoscimento dei segnali in arrivo e al formato richiesto a quelli in uscita. Uno dei parametri più importanti nella comunicazione seriale è la *velocità* di funzionamento di una porta, espressa in *bit al secondo* (*bit per second* o *bps*) e compresa fra 110 e 19200 bps; il numero di byte effettivamente trasmessi al secondo si ottiene dividendo per 10 tale valore, giacché normalmente la trasmissione di un byte richiede 10 bit per la presenza di bit aggiuntivi richiesti dal protocollo. Un altro parametro è il *numero di bit*, 7 o 8, per carattere: infatti se da un lato per trasmettere il set fondamentale di caratteri ASCII sono sufficienti 7 bit, per i caratteri estesi ne servono 8 (si veda l'Appendice A). Altro parametro importante è la *parità*, che definisce l'eventuale metodo di riconoscimento degli errori di trasmissione e che può essere pari, dispari o nessuno. Infine vi è ancora il *bit di stop*, che indica quanti bit, 1 o 2, devono essere posti all'inizio e alla fine di ciascun carattere per separarlo dagli altri. Questi sono solo alcuni dei parametri da regolare, ma già da queste poche notizie si evince come essi vadano accuratamente definiti per poter permettere la comunicazione. Le porte seriali vengono utilizzate per collegare al calcolatore stampanti opportunamente predisposte, plotter, alcuni tipi di mouse, nonché un gran numero di strumenti di misura e apparecchiature scientifiche. Anche i modem utilizzano un collegamento seriale: questi apparecchi modulano i bit provenienti dal calcolatore in suoni e demodulano tali suoni per ritradurli in bit, consentendo così la trasmissione di dati su normali linee telefoniche.

2.3.5 Il bus

Le varie componenti di un calcolatore, il microprocessore, le memorie e gli svariati controller, sono connessi fra loro da un insieme di conduttori paralleli, il *bus*, che permettono lo scambio fra loro dei segnali. Il bus sta alla base dei sistemi ad architettura aperta: se infatti i collegamenti fossero realizzati da cavi ciascuno preposto alla connessione di due specifici elementi, la rete di comunicazione risulterebbe intricata da realizzare e da gestire. Inoltre le possibilità di comunicazione sarebbero strettamente vincolate ai collegamenti iniziali, il che renderebbe complicata l'installazione successiva di nuovi elementi. Si avrebbe quindi un sistema ad architettura "chiusa", fissata una volta per tutte al momento della sua costruzione. Il bus è una rete di collegamenti comuni a tutte le componenti mediante la quale i segnali possono viaggiare da una parte all'altra del calcolatore; esso permette di realizzare sistemi "aperti" non vincolati agli elementi installati al momento della loro costruzione: nuovi componenti possono essere aggiunti in qualsiasi momento inserendoli in apposite *slot di espansione* tutte collegate tramite il bus. Ciò consente all'utente di impiegare innumerevoli tipi e combinazioni di schede opzionali: queste sono dotate di un certo numero di contatti elettrici (*pin*) che realizzano il collegamento della scheda col bus.

I conduttori elettrici del bus, detti *linee*, possono essere suddivisi in tre gruppi, a seconda che vengano usati per trasmettere dati o indirizzi o per controllo; alcune linee sono poi riservate per fornire l'alimentazione, la terra e a distribuire alcuni segnali chiave. Ogni linea trasporta un segnale logico che è interpretato come singolo bit. Il *bus degli indirizzi* (*address bus*) è costituito da tutte le linee che trasmettono gli indirizzi: se per esempio un bus ha 32 linee si possono creare indirizzi con un massimo di 32 bit e quindi il processore può direttamente raggiungere 2^{32} indirizzi. Analogamente il *bus dei dati* (*data bus*) è l'insieme delle linee che trasportano i dati, e il loro numero determina quanti bit possono essere trasmessi simultaneamente: ad esempio un bus a 16 linee trasmette 2 byte alla volta. Fra le linee che costituiscono il *bus di controllo* (*control bus*) vi sono quelle che trasmettono le richieste di interrupt e, in taluni bus, alcune che consentono di contattare ogni singola slot. Infine tra le linee riservate ve ne una che trasmette il segnale del clock.

Le comunicazioni fra componenti avvengono secondo una relazione di *master/slave* (cioè *principale/secondario*): quando un elemento deve effettuare una trasmissione o una richiesta riserva per sé il bus e diventa il master, quindi invia la richiesta all'elemento destinatario che perciò diviene il suo slave, infine rilascia il bus. Talora lo slave invia una risposta al suo master, altre volte dà seguito alla sua richiesta silenziosamente. Ad ogni istante ci può essere un solo master ma sono possibili più di uno slave. Un esempio tipico è il trasferimento di un dato dal processore alla memoria: il processore presenta sulle linee del bus degli indirizzi l'indirizzo di memoria in cui il dato dovrà essere salvato e sul bus dei dati il valore da memorizzare, quindi usa le linee di controllo per richiedere alla memoria la sua registrazione; la memoria a sua volta legge il dato e l'indirizzo richiesto dai relativi bus ed eventualmente usa una delle linee di controllo per avvisare il processore dell'avvenuta memorizzazione. Analogamente avviene la comunicazione fra il processore e le altre periferiche. Quando un elemento non è né master né slave non deve interferire con le trasmissioni in corso sul bus; per questo si usano buffer a tre stati per le uscite: ogni uscita può essere a 0, a 1 o sconnessa dal bus. Per gli ingressi non è strettamente necessario, giacché ogni elemento è libero di "ascoltare" i suoi ingressi anche quando non è coinvolto nelle comunicazioni; poiché però in realtà ciò comporta comunque un assorbimento di corrente dalle linee, solitamente anche gli ingressi vengono sconnessi quando non servono.

Il bus non è solo l'insieme delle connessioni ma anche il protocollo che le governa. Numerosi standard sono stati definiti dall'IEEE e da altre organizzazioni per consentire l'interconnessione di periferiche di Costruttori diversi. Uno dei più diffusi è l'interfaccia *PCI* (*Peripheral Component Interconnect*): esso permette ottime prestazioni usando una frequenza di clock più alta, un bus di dati a 32 bit e un bus di indirizzi a 32 o a 64 bit; inoltre consente una "auto–rivelazione" da parte delle schede inserite, che possono così essere automaticamente configurate all'accensione dal BIOS del bus senza interventi da parte dell'utente, e questo anche grazie al fatto che le schede che aderiscono a questo standard posseggono delle memorie incorporate in cui vengono registrate le informazioni che le riguardano. Il PCI fu disegnato per rimpiazzare lo standard *ISA* (*Industry Standard Architecture*), creato ai tempi dei primi processori Intel ma oramai quasi caduto in disuso: il suo ridotto numero di linee non consente di indirizzare direttamente più di 1 MB di memoria alla volta e inoltre ha un numero limitato di porte logiche e di linee di interrupt (che per di più non sono condivisibili, come invece avviene nello standard PCI); oltre a ciò le schede ISA devono essere configurate dall'utente mediante il posizionamento di microinterruttori presenti su di loro. Lo standard *EISA* (*Extended ISA*) fu pensato per risolvere parte dei problemi connessi all'interfaccia ISA, tra l'altro venendo dotato di 32 linee di indirizzi e della possibilità di ospitare schede senza microinterruttori. Altre interfacce meno diffuse sono lo standard *MCA* (*Micro Channel Architecture*), che sviluppato dalla IBM per i calcolatori PS/2 e alcuni portatili presenta taluni vantaggi rispetto all'ISA/EISA ma non raggiunge le prestazioni del PCI, l'interfaccia *VLB* (*VESA Local Bus*), che però è per certi versi ancor più limitato dell'ISA e comunque non ha grande diffusione, e lo standard *Sbus* usato generalmente sui calcolatori Sparc prodotti dalla SUN.

2.3.6 Trap e interrupt

Trap e interrupt sono due metodi per inviare segnali al processore e richiedere il suo intervento. In entrambi i casi il flusso del programma in corso viene interrotto e il controllo passato ad una specifica procedura dipendente dalla richiesta.

Un *trap* è causato dal sorgere di una condizione anomala durante l'esecuzione di un programma. Un tipico esempio è l'overflow: se il risultato di una operazione matematica eccede in valore assoluto il più grande numero che può essere rappresentato, viene generato un segnale di trap, al che la CPU interrompe il programma in corso e al suo posto esegue una procedura detta *overflow trap handler* (cioè gestore del trap di overflow) che intraprende le azioni più opportune, come stampare un messaggio di errore o registrare il punto nel programma in cui si è verificato. Il punto essenziale quindi è che un trap ha luogo da una condizione anormale generata dal programma stesso e rilevata dall'hardware o dal microcodice della CPU: questo solleva il programmatore dal dover esplicitamente inserire delle istruzioni nei punti critici per intercettare tali condizioni anomale, e di conseguenza consente un risparmio di memoria ed una più alta velocità di esecuzione. Quando un trap viene rilevato, l'indirizzo di memoria dove si trova la corrispondente procedura di gestione viene caricato nel program counter in modo che la CPU la esegua. Oltre al citato overflow, che può riguardare sia numeri interi che reali, altre condizioni comuni che possono generare un trap sono l'underflow, cioè il risultato di una operazione in valore assoluto minore del più piccolo numero rappresentabile, una violazione di protezione, una istruzione in linguaggio macchina inesistente o incorretta, il tentativo di accesso ad una periferica inesistente, e la divisione per zero.

Un *interrupt* è analogo al trap ma è generato non dal programma in esecuzione bensì da qualcosa di esterno, generalmente una periferica di I/O. Un tipico esempio è il trasferimento di dati su disco: la CPU invia una richiesta al disco per poi proseguire nella computazione, e il disco utilizza un interrupt per segnalare il completamento dell'operazione. Contrariamente al trap l'interrupt sospende solo l'esecuzione del programma in corso e trasferisce temporaneamente il controllo ad un opportuno gestore di interrupt (*interrupt handler*), che si prende cura della richiesta fatta. Alla fine il controllo viene restituito al programma sospeso: pertanto è necessario che tale procedura di gestione ripristini la CPU e i suoi registri nello stato in cui si trovavano immediatamente prima che l'interrupt avesse luogo.

La differenza essenziale fra i trap e gli interrupt è che i primi sono sincroni col programma e i secondi sono asincroni: se un programma viene eseguito innumerevoli volte con gli stessi dati iniziali, i trap si verificheranno sempre negli stessi punti, mentre gli interrupt varieranno, dipendendo ad esempio da quanto è occupato il disco o da quando esattamente viene premuto un tasto. E questo perché i trap sono generati direttamente dal programma, gli interrupt solo indirettamente o addirittura indipendentemente.

La necessità degli interrupt nasce dalla possibilità di effettuare l'ingresso e l'uscita in parallelo all'esecuzione dei programmi. Se, dopo aver inviato una richiesta di I/O, la CPU dovesse attendere il suo completamento prima di procedere, si sprecherebbe del tempo prezioso, giacché durante questa attesa potrebbero essere eseguite un gran numero di istruzioni (a volte anche migliaia). Una soluzione alternativa potrebbe essere quella in cui a intervalli regolari la CPU interroga tutte le periferiche per conoscerne lo stato (tecnica nota come *polling*), ma è evidente che anch'essa comporta perdite di tempo (senza contare che l'introduzione di una nuova periferica potrebbe richiedere la riscrittura di parti del sistema operativo). E' da notare che storicamente i primi calcolatori nacquero senza interrupt e che passò molto tempo prima che questa tecnica si affermasse e si diffondesse.

Ad ogni numero di interrupt è associato uno specifico programma di gestione che svolge le operazioni richieste. Gli indirizzi di memoria in cui si trovano tali programmi sono detti *vettori di interrupt* e sono conservati in una apposita tabella, che associa ad ogni interrupt un indirizzo: il sistema se ne serve per determinare la posizione in memoria della procedura che deve occuparsi di gestire la richiesta legata ad uno specifico interrupt (nei calcolatori più vecchi esisteva un unico gestore di interrupt generico che determinava il servizio richiesto leggendo in un apposito registro un codice fornito dalla periferica stessa). Generalmente tali procedure sono contenute nel BIOS, ma possono pure far parte del sistema operativo.

Poiché al termine della gestione dell'interrupt il programma che era in corso deve poter riprendere come se nulla fosse avvenuto, la prima cosa che la procedura deve compiere è salvare il contenuto di tutti i registri della CPU in modo che alla fine possa ripristinarli al loro stato iniziale. A questo scopo si serve dello *stack*: si tratta di una zona riservata di memoria in cui i dati possono essere scritti uno alla volta e riletti in verso opposto. Essi vi vengono "impilati" cioè scritti in sequenza uno dopo l'altro (operazione nota come *stack push*), e vengono poi letti dall'ultimo al primo (*stack pop*)*.

La procedura di gestione di un interrupt è un programma come gli altri, e può essere a sua volta interrotto da un interrupt successivo: se ciò avviene, la nuova procedura prende il posto della vecchia, a sua volta sospesa, salva anch'essa lo stato dei registri, che verranno posti in locazioni dello stack successive alle precedenti, svolge il suo compito, e infine fa riprendere il lavoro alla prima procedura. Ad esempio, nulla vieta che durante la gestione dell'interrupt dal disco arrivi un altro interrupt dalla tastiera o dalla scheda di rete. Può così accadere che ad un certo istante più interrupt siano "annidati" uno nell'altro, con l'ultimo in esecuzione e i precedenti sospesi. Affinché questo meccanismo funzioni senza intoppi le procedure di gestione devono possedere due caratteristiche. Innanzitutto devono essere *trasparenti*, cioè il programma che viene interrotto non deve accorgersi della loro presenza: ecco perché alla fine lo stato della CPU deve essere ripristinato come era subito prima dell'interruzione. Inoltre esse devono essere *rientranti*, cioè non devono basarsi sul contenuto di zone di memoria usate come aree di lavoro da altri interrupt e che quindi potrebbero venir modificate se si verificasse una richiesta di interrupt durante la loro esecuzione. Se un calcolatore gestisce molte periferiche una procedura di interrupt può essere sospesa e ripresa più volte prima di essere completata.

Lo scenario reale non è tuttavia così semplice: i programmi di gestione degli interrupt non devono essere sospesi mentre stanno svolgendo operazioni cruciali. Ad esempio è bene non interromperne l'esecuzione durante il salvataggio iniziale dei registri nello stack o durante il loro recupero finale. Ci sono poi delle procedure critiche che vanno completate

---

* Questa tecnica di memorizzazione viene indicata come *LIFO* ossia *Last In First Out*, perché l'ultimo dato ad essere inserito è anche il primo ad essere scaricato; la tecnica in cui i dati vengono invece letti nello stesso ordine in cui sono stati scritti è detta *FIFO* per *First In First Out*.

entro un certo tempo: se per esempio un modem riceve 1000 caratteri al secondo, il programma che lo gestisce deve leggere e salvare un carattere entro 1 ms, altrimenti questo sarà sovrascritto dal successivo e andrà perso. In altre parole quando ha inizio un'operazione più importante, questa non deve essere interrotta da richieste meno importanti. Ciò si ottiene *disabilitando* temporaneamente gli interrupt. A questo scopo i processori più semplici hanno un solo bit (generalmente contenuto nella program status word): quando esso è posto a 0, il che si ottiene con una apposita istruzione in linguaggio macchina, la stragrande maggioranza degli interrupt (i cosiddetti *interrupt mascherabili*) è disabilitata, e solo pochissimi interrupt critici (gli *interrupt non mascherabili*, di solito procedure legate al funzionamento essenziale del calcolatore) possono aver luogo comunque. Quando la procedura ha completato la parte critica riabilita gli interrupt ponendo a 1 il suddetto bit per mezzo di un'altra istruzione in linguaggio macchina. I processori più evoluti possiedono non uno ma un'intera maschera di bit, ognuno associato a singoli o gruppi di interrupt: ogni procedura di gestione scrivendo nella maschera determina quali richieste la possano interrompere, ponendo i relativi bit a 1, e quali invece debbano attendere. Processori ancor più sofisticati possiedono un registro dedicato in cui è immagazzinata la priorità assegnata all'interrupt servito in quel momento: il relativo programma può essere sospeso solo da interrupt con priorità uguale o più alta. Questa priorità come pure la disabilitazione degli interrupt è decisa dal programmatore all'atto della creazione delle procedure di gestione, stabilendo quali siano le più importanti e quali le meno critiche al fine di ottimizzare il funzionamento del calcolatore. In tutti e tre i casi comunque se si verifica un interrupt fra quelli momentaneamente disabilitati, esso non va certo perso ma viene tenuto in sospeso: la CPU se ne occuperà appena l'esecuzione delle parti critiche di codice sarà terminata.

Quelli trattati finora costituiscono in realtà uno dei due tipi di interrupt possibili, detti *interrupt hardware* perché generati dalle periferiche del calcolatore. Esistono però anche gli *interrupt software* azionati dal programma in corso per mezzo di una istruzione in linguaggio macchina. In questo modo un programma può utilizzare per proprî fini talune procedure del BIOS, il che consente ad un programmatore esperto di evitare di dover riscrivere segmenti di codice magari meno efficiente. Ad esempio, premendo il tasto *Stampa* (o *Print Screen*), che determina la stampa della schermata video attuale, di fatto viene generato uno specifico interrupt, la cui procedura di gestione effettua materialmente l'operazione di stampa: allora se un programma necessita di effettuare una stampa, un modo molto semplice è di richiedere lo stesso interrupt. Ma c'è di più: poiché molti numeri di interrupt sono liberi, cioè non associati ad alcun elemento hardware, è possibile scrivere un sottoprogramma utente, caricarne l'indirizzo di memoria nella tabella dei vettori di interrupt in una posizione libera, e quindi richiamarlo quando serve attivando via software il corrispondente interrupt. Vi è una differenza sostanziale fra i trap e gli interrupt software: un trap è generato per così dire "involontariamente" dal programma in esecuzione a causa del verificarsi di una condizione di errore o comunque anomala non voluta o non prevista (a meno di non pensare che il programmatore abbia deliberatamente inserito una divisione per zero o una istruzione inesistente!), e quasi sempre di una tale gravità da determinare la fine prematura del programma; l'interrupt software invece è attivato "volontariamente" dal programma e, a meno di errori gravi di programmazione, alla sua conclusione quest'ultimo riprende il suo corso.

Bibliografia

A. S. Tanenbaum, *Structured Computer Organization*, ed. Prentice–Hall, 1984

D. D. Swade, *La ricostruzione della macchina da calcolo di Charles Babbage*, in *Le Scienze*, n. 297, Mag. 1993, p. 40

S. Hénin, M. Zaninelli, *Il calcolo automatico negli Stati Uniti, dalle origini al 1950*, in *Le Scienze*, n. 344, Apr. 1997, p. 64

Those parts of the system that you can hit with a hammer
are called hardware;
those program instructions that you can only curse at
are called software.

Levitating Trains and Kamikaze Genes: Technological Literacy for the 1990's.

Capitolo III

I sistemi operativi

An elephant is a mouse with an operating system.

Linux “fortune”

§ 3.1 Files e cartelle

3.1.1 I files

Le informazioni registrate su disco, testi, dati, programmi e così via, sono raggruppate in insiemi logici detti *files** : un file è una collezione di dati che può essere trattata come una singola entità, come appunto un programma, sorgente o eseguibile, il testo di un documento o l’insieme dei dati di un apparato raccolti in un certo tempo.

I dati di un file sono sistemati in una sequenza di *record logici* (*logical records*), ognuno dei quali rappresenta una qualche unità di informazione. Nei casi più semplici può trattarsi anche di un singolo carattere o valore numerico, oppure di ciascuna riga di un file di testo; in generale però dipende dall’applicazione che ha creato il file o che lo usa: se ad esempio un file contiene i dati raccolti da un esperimento, un record logico può essere la struttura costituita dalle informazioni relative ad un singolo evento, come l’ora a cui si è verificato e il contenuto delle memorie dei moduli elettronici di lettura interessati. I record di un file possono avere lunghezza fissa uguale per tutti (*fixed–length records*) oppure lunghezza variabile (*variable–length records*).

Prima di poter essere letto o scritto un file deve essere “aperto” dal programma che lo vuole utilizzare: questa operazione rende noto al sistema di gestione dei files che un

* Letteralmente l’inglese *file* significa *archivio*, *documento*, ma nessuno di questi termini italiani rende bene il significato particolare che esso assume nell’informatica, per cui è più conveniente usare la voce inglese.

programma utente desidera farne uso, cosicché esso può assegnare delle risorse che servono a localizzarlo fisicamente sul disco; inoltre in caso di lettura controlla che il file esista e che l'utente sia autorizzato ad accedervi, mentre in caso di scrittura a volte prenota lo spazio necessario. Quando si è terminato di operare su detto file, è opportuno "chiuderlo": con ciò vengono rilasciate le risorse che erano state precedentemente riservate, e nel caso di file aperto in scrittura le eventuali modifiche vengono rese permanenti sul disco. Tutti i linguaggi di programmazione dispongono per questo di *istruzioni di apertura* (*open instructions*) e *di chiusura* (*close instructions*).

Una volta aperto il file è possibile richiedere al sistema la lettura o la scrittura di un record. I file possono essere *ad accesso sequenziale* (*sequential access files*) o *ad accesso diretto* o *casuale* (*direct access* o *random access files*). Nel primo caso ad ogni lettura si ottiene il record seguente: il sistema tiene conto di dove è arrivato a leggere, e alla richiesta successiva fornisce il record successivo, cominciando sempre dal primo. Così se l'utente vuole sapere il contenuto del record n deve prima leggere gli $n-1$ che lo precedono. Nel secondo caso invece il programma può leggere immediatamente il record n che gli interessa passandone il numero al sistema di gestione attraverso l'istruzione di lettura o di scrittura. Ciò è utile quando si vogliano leggere i record di un file in un ordine diverso da come sono stati scritti, oppure quando se ne voglia modificare il contenuto di uno solo senza alterare gli altri. Il tipo di accesso consentito al file, diretto o sequenziale, viene deciso al momento della sua creazione. In genere la scelta del record in un file ad accesso casuale viene fatta per mezzo del suo numero d'ordine; in alcuni casi invece si può indicare attraverso il contenuto di un qualche campo, detto *chiave* o *key*, all'interno del record stesso: se ad esempio un file ad accesso diretto contiene i dati anagrafici di un gruppo di persone, potrebbe essere possibile richiamare i record in base al campo che contiene la loro data di nascita. Ovviamente l'accesso diretto ai record di un file è possibile solo su quei supporti, come dischi e dischetti, che a loro volta sono ad accesso casuale: sui supporti ad accesso sequenziale, come i nastri magnetici, sono possibili solo file ad accesso sequenziale.

Infine è da notare che dal punto di vista pratico, i files si possono ulteriormente suddividere in *formattati* e *non formattati* (*formatted* e *unformatted*): nei primi i dati, in genere costituiti da semplici caratteri alfanumerici, hanno un formato di stampa e sono direttamente esaminabili e modificabili dall'utente (ne sono esempi i file di testo o i programmi sorgente), nei secondi invece tale formato è assente e risulta molto più conveniente manipolarli attraverso i programmi che li hanno creati (ne sono un esempio i programmi eseguibili).

3.1.2 Le directory

L'insieme dei dati che costituiscono un file vengono registrati sui settori del disco (come si vedrà meglio in seguito trattando della gestione dello spazio), e per poterli recuperare il sistema memorizza il punto di inizio di ciascuno. Per un utente tuttavia è di certo scomodo ricordare il contenuto di un file o il suo uso in base ad un numero o ad un codice, e pertanto tutti i sistemi consentono di assegnargli un nome. Spesso poi è utile conoscerne anche la dimensione in byte e magari la data e l'ora in cui è stato creato o modificato; infine, se il sistema consente la definizione di più utenti, deve essere noto anche l'utente cui il file appartiene ed eventualmente chi altri sia autorizzato a leggerlo, modificarlo o distruggerlo. Tutte queste informazioni, insieme ad altre più specifiche, vengono salvate in un'area di sistema del disco detta *directory radice* o *cartella principale* * : si tratta di un elenco dei files contenuti sul disco e delle relative informazioni. Fra queste vi sono appunto il nome, la dimensione, la data e l'ora in cui è stato creato o modificato l'ultima volta, e la locazione sul disco in cui hanno inizio i dati che lo compongono (per l'esattezza, vi è l'indirizzo del primo blocco, si veda il par. 3.1.3); nei sistemi a molti utenti vi sono anche il proprietario del file, ovvero l'utente che l'ha creato o comunque a cui appartiene, e un elenco di *protezioni*, cioè una serie di dati, il cui formato dipende molto dal sistema di gestione dei files utilizzato, che indicano quali utenti siano abilitati a leggerne il contenuto ed eventualmente a modificarlo o persino rimuoverlo dal disco. A questi dati molti sistemi aggiungono altri attributi usati internamente per tipi particolari di files.

Quando un utente richiede i dati di un file, il sistema ne cerca il nome nella directory, verifica se esiste e se l'utente è autorizzato a leggerlo, e quindi ricava in corrispondenza del nome l'indirizzo della sua locazione di inizio su disco, infine invia a quest'ultimo una richiesta di lettura dei settori corrispondenti. Analogamente, in caso di scrittura, il sistema

* Personalmente l'Autore considera impronunciabile il termine *direttorio*.

dopo aver verificato se l'utente è abilitato per tale operazione invia al disco una richiesta di scrittura e quindi registra nella directory il nome del file e il suo punto di inizio, ovvero li aggiorna in caso di modifica di un file già esistente.

Se esistesse solo la directory radice tutti i files verrebbero registrati in un unico elenco, il che ne renderebbe certamente scomoda l'organizzazione (senza contare che alcuni sistemi, come il DOS, fissano un numero massimo di files che possono trovar posto nella directory radice). Per questo tutti i sistemi consentono di creare delle *subdirectory*. Ognuna di esse è per certi versi un file come gli altri, con un suo nome, una sua data, eventualmente un proprietario e delle protezioni, e naturalmente una sua locazione sul disco, ma il suo contenuto è particolare: infatti analogamente alla directory radice una subdirectory contiene i nomi e le informazioni di altri files su disco. Per poterle distinguere dai files ordinarî il sistema aggiunge un attributo particolare ai dati delle directory. Una subdirectory può ovviamente contenere a sua volta altre subdirectory, creando così un "albero" di directory che consente di suddividere e catalogare facilmente i proprî files; questa situazione è esemplificata in Fig. 3.1 .

All'interno di una directory due files devono necessariamente avere nomi diversi, altrimenti il sistema non saprebbe come distinguerli; è invece possibile dare lo stesso nome a due files in due directory differenti. Al nome di un file si suole aggiungere una *estensione*, in genere una stringa mnemonica separata da un punto, che aiuta l'utente, e a volte anche il sistema operativo, a identificare il contenuto del file (testo, dati, programma sorgente e linguaggio in cui è scritto, e così via). In alcuni ambienti, in Unix ad esempio, l'estensione è parte integrante del nome, in altri, come in DOS e in VMS, l'estensione è invece un attributo a sé registrato in un campo dedicato della struttura di informazioni che la directory conserva per ciascun file. Il VMS inoltre assegna un altro attributo ad ogni file, il *numero di versione* (*version number*), il che di fatto consente di dare lo stesso nome a più files, che vengono però distinti da un numero progressivo.

3.1.3 La gestione dello spazio su disco

Come si è visto al par. 2.3.3, le operazioni di ingresso e uscita di basso livello su disco sono eseguite solo per settori completi. Il sistema di gestione dei files suddivide lo spazio

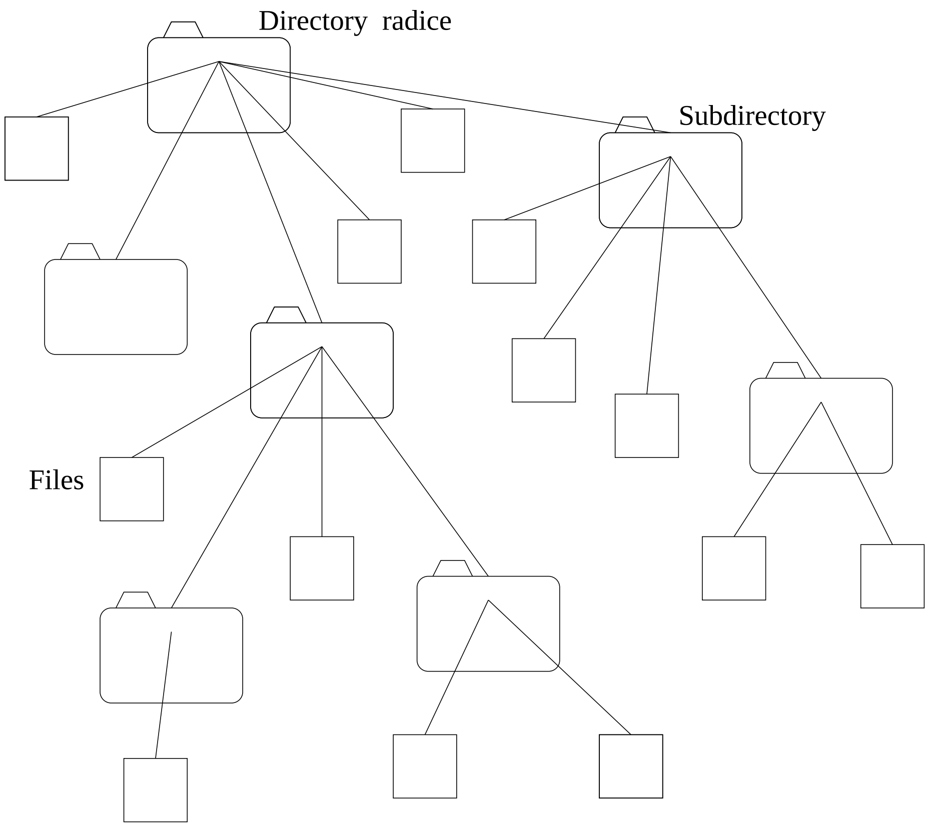

Fig. 3.1 *Un esempio di un albero di directory su disco*

disponibile in unità elementari, dette *cluster* o *blocchi* (*blocks*), ognuna costituita da uno o più settori: lo spazio occupato da ogni file sarà un numero intero di cluster.

Il sistema cerca per quanto possibile di salvare il contenuto di un file in blocchi contigui, poiché questo rende più rapide lettura e scrittura. Quando il disco è quasi completamente vuoto risulta molto semplice trovare una sistemazione in cluster consecutivi; mano a mano che il disco si riempie questa scelta diventa sempre più difficile. Una ulteriore complicazione è data dal fatto che non sempre è nota a priori la dimensione finale: se un programma sta progressivamente riempiendo di dati un file, questo crescerà nel tempo e ad un certo punto non sarà più possibile trovare cluster consecutivi in cui salvarne il contenuto, perché questi sono già occupati da altri files. Si dice allora che il file viene *frammentato*. Quando poi la percentuale di files frammentati diventa alta, le prestazioni del disco si degradano rapidamente, poiché la testina di lettura deve spostarsi di continuo da una parte all'altra del disco. Per questo alcuni sistemi hanno dei programmi che consentono di "deframmentare" il disco, cioè di ridurre per quanto possibile la frazione di files non disposti in cluster

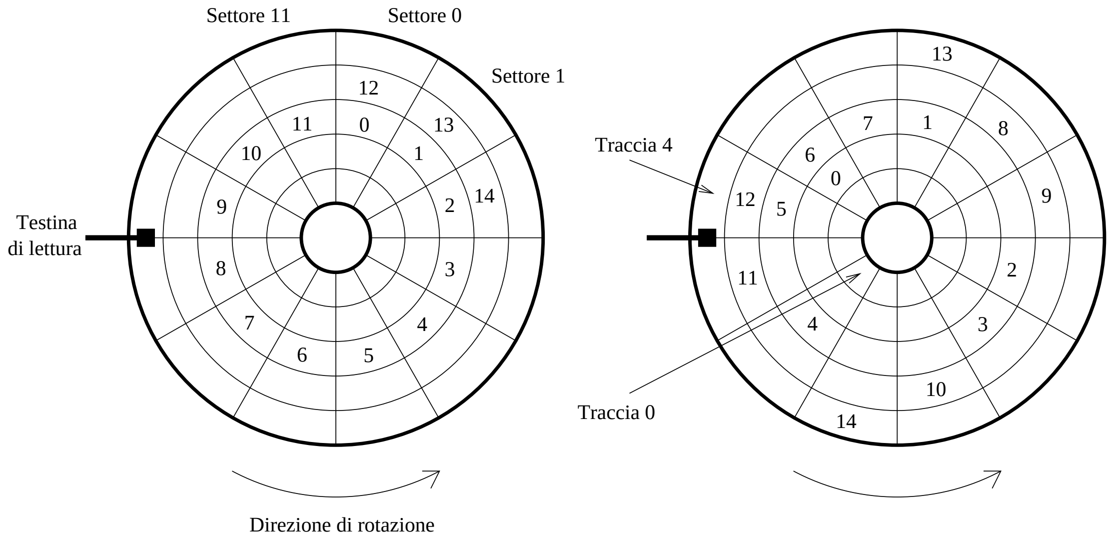

Fig. 3.2 *Due possibili modi di salvare un file su disco: a) in blocchi adiacenti, b) in blocchi non adiacenti*

consecutivi. In Fig. 3.2 sono esemplificate le due situazioni di un file registrato in blocchi adiacenti e dello stesso file frammentato in blocchi non adiacenti, nell'ipotesi semplice che ogni blocco sia costituito da un solo settore.

Per il sistema di gestione un file consiste in una collezione ordinata, anche se non necessariamente consecutiva, di blocchi fisici, mentre ad un programma utente lo stesso file appare come una sequenza lineare di record logici. In generale la dimensione di un record sarà diversa da quella di un cluster: per esempio, i cluster di un disco potrebbero essere di 1kB = 1024 byte l'uno e i record di un certo file invece di 100 byte. Allora i byte da 0 a 99 del primo blocco conterranno il primo record, i byte da 100 a 199 il secondo, e così via. I record sono considerati logicamente consecutivi, anche se sono fisicamente separati in blocchi non contigui e può accadere che un record sia diviso in due blocchi diversi (nell'esempio precedente, l'undicesimo record occupa i byte da 1000 a 1023 del primo blocco e i byte da 0 a 75 del secondo). E' compito del sistema di gestione dei files rendere trasparente al sistema operativo in generale, e ai programmi utente in particolare, l'organizzazione fisica dei files su disco: quando un programma richiede il record n deve ottenere il record n indipendentemente da quale o quali cluster esso occupi. Si noti infine che pochissimi files occuperanno un numero intero di blocchi, e pertanto andrà sprecato un po' di spazio nell'ultimo blocco di quasi tutti i files (se il file dell'esempio precedente è costituito da 30 record, contiene in totale $100 \times 30 = 3000$ byte ma occupa 3 blocchi, pari a 3072 byte, dei quali 72 restano inutilizzati). Quanto più grande è la dimensione del cluster tanto maggiore sarà la percentuale di spazio inutilizzato; quindi se si sa a priori che la dimensione media dei files sarà piccola, è controproducente usare una dimensione dei cluster grande (di solito tale dimensione può essere decisa al momento della formattazione logica; tuttavia alcuni sistemi operativi ne fissano un valore minimo anche in base alla capacità totale del disco).

Per poter ricostruire la posizione sul disco dei dati di un file, il sistema deve conoscere in quali blocchi si trovano. Esistono essenzialmente due metodi per fare ciò. Il sistema può tenere una tabella (*file index*) con la sequenza dei blocchi che costituiscono ciascun file; tale tabella può essere unica per tutti i files (come avviene nel VMS; anche il DOS usa una sola tabella, la *FAT* o *File Allocation Table*, ma la utilizza anche per registrare i blocchi liberi), oppure ve ne può essere una per ogni file (come fa lo Unix). In alternativa ogni blocco può contenere al suo interno l'indirizzo del successivo (*linked list*), il che è un po' equivalente a disperdere il file index lungo tutto il file. Il primo metodo è però più comodo, visto che consente di localizzare rapidamente un dato blocco (e quindi un dato record) senza dover esaminare tutti quelli che lo precedono. In entrambi i casi l'indirizzo del primo blocco è scritto nella directory in corrispondenza di ogni file.

Per poter assegnare lo spazio su disco il sistema di gestione dei files deve anche conoscere quali cluster sono liberi e quali già occupati. A tal fine può impiegare o una lista di blocchi contigui liberi (*free list*), oppure una tabella di bit (*bit map*), con un bit per ogni blocco, dove un 1 indica un cluster occupato e uno 0 un cluster libero. La lista di blocchi ha il vantaggio di semplificare la ricerca di un gruppo di cluster liberi di una certa dimensione, ma ha lo svantaggio di avere una lunghezza variabile che cambia nel tempo come i files vengono creati e cancellati, certamente una scomodità. La tabella di bit presenta invece il vantaggio di avere una lunghezza fissa, determinata dal numero totale di blocchi che è una quantità costante, e inoltre per cambiare lo stato da libero a occupato e viceversa è sufficiente alterare un bit; tuttavia è complicato trovare un'area libera della dimensione voluta. In entrambi i casi queste tabelle vanno aggiornate ogniqualvolta un file venga creato, cancellato, espanso o contratto; e se la lista o la tabella si corrompono o rimangono non sincronizzati con lo stato effettivo dello spazio su disco, l'intero sistema di files può restarne compromesso e si corre il serio rischio di perdere i dati contenuti. Per questo tutti i sistemi operativi controllano periodicamente questo elemento cruciale dei dischi.

Oltre che creato e modificato, un file può essere anche cancellato. Quando ciò avviene però i dati in esso contenuti non vengono rimossi dal disco: il sistema si limita a eliminare ogni riferimento al file dalla directory e a marcare nella apposita lista o tabella di bit come liberi e quindi riutilizzabili i blocchi precedentemente occupati dal file. Teoricamente quindi è possibile recuperare i dati di un file cancellato, purché ovviamente il sistema non vi abbia già riscritto sopra. Per questo l'operazione di recupero è relativamente semplice su quei calcolatori il cui sistema operativo non agisce in maniera autonoma e non consente a più utenti di accedervi contemporaneamente (come il DOS), ma diventa molto ardua in quei sistemi che viceversa possono operare autonomamente e permettono diversi utilizzatori allo stesso tempo (come lo Unix; si veda il paragrafo seguente per i dettagli sui sistemi a singolo utente e multiutente): in questo caso infatti non appena i blocchi si rendono disponibili possono essere usati dal sistema di gestione per salvare i dati di altri utenti o dello stesso sistema operativo, indipendentemente dalla volontà e dalla possibilità di intervento dell'utente che ha cancellato il proprio file.

Lo spazio su disco può essere usato nella sua totalità per la registrazione dei dati. Spesso tuttavia è più conveniente suddividere tale spazio in aree distinte, dette *partizioni*: una partizione è un insieme di blocchi contigui che formando una regione a sé stante viene usata separatamente dalle altre. Infatti i sistemi di gestione dei files di tutti i sistemi operativi rispettano strettamente i confini delle partizioni, pertanto qualunque file non può che trovarsi tutto all'interno di una sola di esse e non può stare a cavallo di due o più. Di fatto ogni partizione si comporta quasi come fosse un disco distinto. Ciò può sembrare una limitazione, giacché può accadere che una partizione sia talmente piena da non permettere il salvataggio di un dato file quando invece un'altra risulta quasi vuota. Tra l'altro il tentativo di registare i dati in blocchi adiacenti e la frammentazione vengono gestiti indipendentemente per ciascuna partizione. Tuttavia questa tecnica torna comoda se per una qualsiasi ragione si vogliono tenere fisicamente separati gruppi di dati differenti, e diventa indispensabile quando devono convivere due o più sistemi operativi diversi, visto

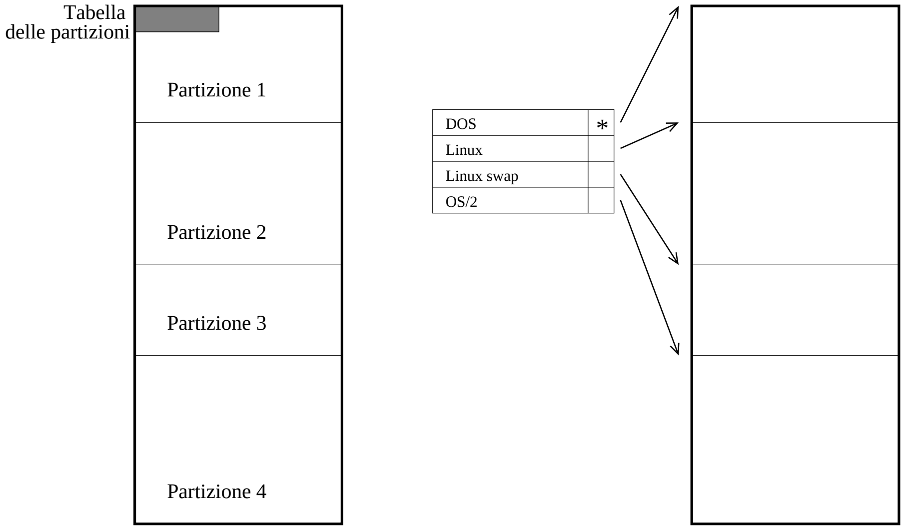

Fig. 3.3 *Schema dell'uso della tabella delle partizioni per suddividere un disco fisso*

che ognuno ha un proprio modo di organizzare e registrare i dati su disco, generalmente incompatibile con quello degli altri: infatti all'interno di una partizione è possibile effettuare una formattazione logica* in grado di soddisfare le esigenze di ogni specifico sistema di gestione dei files. Questo meccanismo offre una grande flessibilità nell'uso dei dischi fissi, permettendo la coesistenza di sistemi operativi diversi. Vi è comunque un prezzo da pagare: una volta create le partizioni, le loro dimensioni non possono essere cambiate senza distruggere i dati in esse contenuti, ed è facilmente comprensibile la ragione, in quanto i sistemi di gestione in generale utilizzano tutto lo spazio di una partizione per registrare i dati dei files; se parte di questo spazio viene assegnato ad un'altra partizione, il sistema non è in grado di garantire la consistenza della sua struttura interna, e pertanto sarà necessario eseguire da capo la formattazione delle nuove partizioni.

Per effettuare questa suddivisione il primo settore di un disco fisso viene riservato per contenere una *tabella delle partizioni* (*partition table*) con le informazioni necessarie a definire ogni singola partizione: in particolare questa tabella contiene l'indirizzo di inizio e fine della stessa, dati in numero di cilindro, un codice indicante il tipo di partizione e se si tratta di una partizione "attiva", cioè se può essere usata per caricare un sistema operativo. In Fig. 3.3 è mostrato schematicamente l'uso della tabella delle partizioni, mentre in Tab. 3.1 sono elencati gli identificatori dei più diffusi tipi; tale identificatore è

* La formattazione fisica invece non può che essere la stessa per l'intero disco.

utilizzato da molti sistemi operativi per riconoscere il tipo di sistema di gestione dei files installato nella rispettiva partizione. Tecnicamente è ammesso che le partizioni siano non confinanti fra loro, cioè che vi siano dei cilindri non appartenenti ad alcuna partizione, ma questo non è consigliabile perché tali cilindri non sono accessibili e quindi vanno sprecati.

Tabella 3.1 *Principali tipi di partizione e loro identificatori esadecimali.*

Partizione	Codice	Partizione	Codice
Vuota	00	Novell Netware 386	65
DOS 12-bit FAT	01	PIC/IX	75
XENIX root	02	Old MINIX	80
XENIX usr	03	Linux/MINUX	81
DOS 16-bit <=32M	04	Linux swap	82
Estesa	05	Linux nativa	83
DOS 16-bit >=32M	06	Linux estesa	85
OS/2 HPFS	07	Amoeba	93
AIX	08	Amoeba BBT	94
AIX attiva	09	BSD/386	A5
OS/2 Boot Manager	0A	OpenBSD	A6
Win95 FAT32	0B	NEXTSTEP	A7
Win95 FAT32 (LBA)	0C	BSDI fs	B7
Win95 FAT16 (LBA)	0E	BSDI swap	B8
Win95 Estesa (LBA)	0F	Syrinx	C7
Venix 80286	40	CP/M	DB
Novell	51	DOS access	E1
Microport	52	DOS R/O	E3
GNU HURD	63	DOS secondary	F2
Novell Netware 286	64	BBT	FF

Un disco, sia di tipo IDE che di tipo SCSI, può contenere da un minimo di una partizione, che nel caso più semplice occuperà l'intera superficie, ad un massimo di quattro: queste partizioni, la cui descrizione è contenuta nella suddetta tabella, sono dette *partizioni primarie* (*primary partitions*). Come si vede nella Tab. 3.1 esiste un tipo particolare di partizione primaria detta *partizione estesa* (*extended partition*), che si comporta quasi fosse un disco a sé stante: insieme ad una partizione estesa infatti viene creata pure una *tabella di partizione estesa*, e per questo essa può contenere una o più *partizioni logiche* (*logical partitions*, cosiddette per distinguerle dalle primarie). In Fig. 3.4 è mostrato un esempio di disco con una partizione primaria ed una estesa a sua volta divisa in due partizioni logiche più dello spazio libero non partizionato. Mentre vi possono essere al massimo

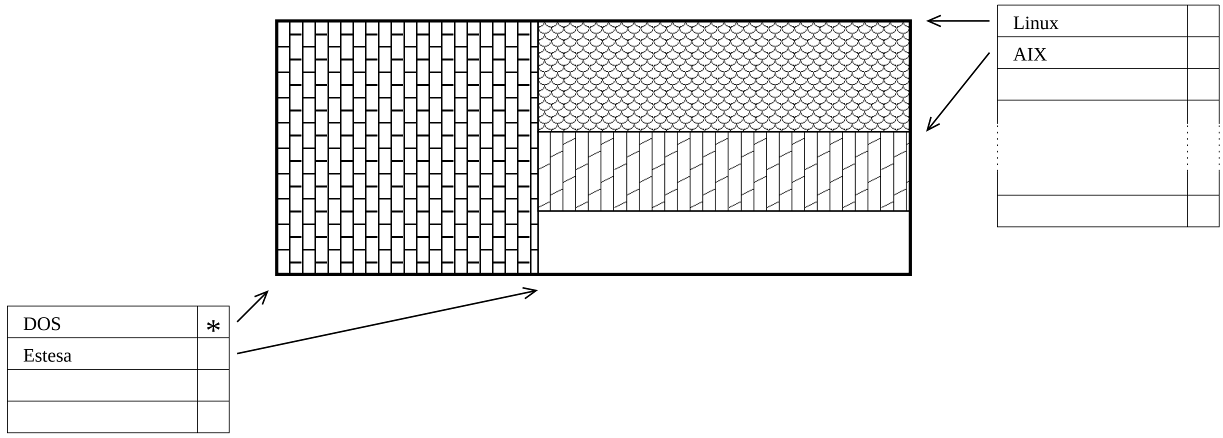

Fig. 3.4 *Schema di un disco con una partizione primaria ed una estesa contenente due partizioni logiche*

quattro partizioni primarie, non vi è limite al numero di partizioni logiche che possono esistere su un disco (anche se molti sistemi operativi hanno delle difficoltà a gestirne più di dodici).

§ 3.2 Generalità sui sistemi operativi

Un calcolatore senza un programma da eseguire è poco più che un ammasso di componenti elettronici che producono calore: ciò che lo rende utile ed efficace sono i programmi. Nei primi calcolatori tuttavia il programma era determinato dal cablaggio dei suoi componenti e di conseguenza qualsiasi variazione era lunga e difficoltosa. Nei calcolatori moderni invece i programmi vengono registrati nella memoria, volatile o permanente, nello stesso modo in cui lo sono i dati, pertanto ogni successivo cambiamento da parte dell'utente è rapido e semplice, e questo li rende molto più flessibili e adattabili a scopi diversissimi fra loro.

Il *sistema operativo* è l'insieme dei programmi che rendono servibile il calcolatore. Alcuni di questi programmi vengono eseguiti autonomamente dal calcolatore stesso, alla partenza, ad intervalli regolari o quando necessario, per assicurarne il corretto funzionamento; altri programmi invece sono esplicitamente invocati dall'utente per interagire con i componenti del calcolatore e quindi utilizzarlo per svolgere il proprio lavoro. Del sistema operativo fanno poi parte anche files di configurazione che forniscono ai programmi di sistema i dati necessarî per una corretta esecuzione.

Affinché il calcolatore possa svolgere le sue funzioni è necessario che vi sia un programma particolare che ne supervisioni il funzionamento; esso deve entrare in azione all'accensione e rimanere attivo per tutto il tempo in cui il calcolatore viene usato. Tale programma particolare è il *nucleo*, o *kernel*, del sistema operativo. Le principali funzioni che esso deve svolgere sono

- la gestione della unità di elaborazione centrale, decidendo quali programmi debbano essere eseguiti, in che ordine, con quale priorità e per quanto tempo;
- la gestione della memoria volatile, quali programmi e dati o parti di essi vadano caricati in memoria e in quali zone, e quali altri vadano invece definitivamente cancellati o provvisoriamente trasferiti per esigenze di spazio nella memoria di massa;
- la gestione dei files e la loro manipolazione da parte dell'utente senza che questi debba conoscere la struttura del loro sistema di gestione, e garantendo che le informazioni memorizzate siano accessibili solo dagli utenti autorizzati;
- la gestione delle operazioni di ingresso e uscita, in modo da rendere trasparente all'utente, almeno in parte, la complessità delle operazioni hardware e la conoscenza della struttura fisica e del funzionamento delle periferiche;
- la gestione degli errori e dei malfunzionamenti, per evitare danni ai dati contenuti nelle memorie, volatile e di massa, e garantire per quanto possibile la continuità di funzionamento.

Inoltre il kernel raccoglie informazioni statistiche sul funzionamento del sistema, assegna le risorse ai varî programmi da eseguire e le rilascia al termine, riconosce gli utenti abilitati a svolgere determinate operazioni o ad accedere a determinate informazioni, e nei sistemi a pagamento svolge le funzioni di addebito del tempo macchina utilizzato da ogni utente.

Il sistema operativo deve essere protetto dalla possibilità che i programmi di un utente agiscano in zone della memoria riservate al sistema o ad altri utenti. Il kernel si preoccupa anche di gestire le risorse (CPU, memoria, dischi, periferiche varie) quando più programmi sono in esecuzione per evitare che si intralcino fra loro e al tempo stesso per ottimizzare le prestazioni del sistema.

Si possono distinguere diverse modalità di elaborazione a seconda di come il sistema operativo gestisce le risorse del calcolatore. La più semplice è la *macchina dedicata uniprogrammata*, che può essere utilizzata da un solo utente alla volta ed esegue un solo programma alla volta. In questo caso il sistema operativo è molto semplice e fornisce solo alcune funzioni di gestione dei files, di ingresso e uscita e di preparazione dei programmi. E' un tipo di sistema operativo ancora utilizzato, anche se quasi esclusivamente su piccole macchine giacché non consente di sfruttare appieno le risorse a disposizione. Un esempio è costituito dal DOS: in questo caso alla partenza non viene avviato un vero e proprio kernel, ma piuttosto un interprete di comandi, che esegue uno alla volta quelli impartiti dall'utente; durante l'esecuzione di un programma tutta la CPU e tutta la memoria sono a disposizione dell'utente, e il sistema rimane impegnato finché tale esecuzione è terminata. Poiché non vi è concorrenza fra programmi, non ci sono meccanismi di protezione della memoria, e l'utente può accedere direttamente in lettura e scrittura a qualunque locazione. Essendoci un solo utente non esistono meccanismi di autenticazione, né vi è protezione tra i files, che appartengono tutti all'unico utilizzatore, col rischio però che files e dati di sistema vengano modificati e potenzialmente danneggiati dall'azione dell'utente. Più evoluta è la *macchina dedicata multiprogrammata*, che pur potendo essere utilizzata da un solo utente per volta è in grado di eseguire più compiti allo stesso tempo (*multitasking*). In questo caso esiste un vero kernel che si preoccupa di gestire le risorse assegnandole con priorità diverse ai diversi programmi per evitare conflitti; per queste ragioni tali risorse, e la memoria in particolare, non sono accessibili direttamente dai programmi utente. Il sistema operativo è più complesso e comprende programmi di gestione che vengono attivati all'occorrenza dal sistema stesso. E' possibile sospendere un programma per eseguirne un altro, oppure mandarne più di uno in esecuzione. Nel caso in cui la memoria fisica non fosse sufficiente, può intervenire un meccanismo detto *memoria virtuale* in grado di espanderne la capacità in modo da assegnare ad ogni programma una quantità sufficiente di memoria di lavoro. Inoltre è possibile che più utenti diversi possano usare il calcolatore, ancorché uno alla volta, e per questo vengono previsti meccanismi di autenticazione e di protezione dei dati di sistema e dei singoli utenti. Esempi tipici sono MicroSoft Windows e Apple MacIntosh. Ancor più sofisticata è la *macchina multiutente*, che può essere utilizzata da più utenti allo stesso tempo. Necessariamente si fa uso della multiprogrammazione; ogni utente deve essere autenticato dal sistema e può proteggere in tutto o in parte i suoi files personali dagli altri utilizzatori; gli stessi files di sistema sono protetti e possono essere manipolati solo dagli utenti autorizzati. Salvo casi particolari ciascun utente può eseguire più programmi, e questi di solito possono proseguire il proprio lavoro anche quando lo stesso non è più collegato al sistema; per questo una sufficiente quantità di memoria virtuale viene assegnata ad ogni utente. Spesso è possibile l'accesso remoto attraverso altri calcolatori per mezzo di una rete di comunicazione. Molti sono i sistemi operativi multiutente, i più noti sono lo Unix, di cui si tratterà diffusamente in seguito, e il VMS. Un caso particolare poi è costituito dai sistemi *in tempo reale* (*real–time systems*): si tratta di sistemi di elaborazione delle informazioni in grado di rispondere in un tempo trascurabilmente piccolo a stimoli esterni. Un esempio classico sono i sistemi di acquisizione dati di esperimenti scientifici, i quali debbono essere capaci di elaborare le informazioni ricevute ed eventualmente salvarle minimizzando i tempi morti; altri esempi sono i sistemi di controllo dei processi industriali, nonché i sistemi centralizzati di elaborazione dati ai quali si può accedere da molti terminali sparsi in varie zone, come ad esempio i calcolatori centrali delle banche o delle compagnie aeree, i quali devono rispondere rapidamente alle richieste per mantenere aggiornati, appunto in tempo reale, i loro archivi.

I calcolatori possono essere dotati di varie CPU in grado di lavorare simultaneamente (*sistemi a multiprocessore*): in questo caso più programmi sono effettivamente eseguiti in parallelo. La maggior parte dei calcolatori però è dotata di una sola unità di elaborazione, quindi anche nei sistemi a multiprogrammazione in realtà ad ogni istante un solo programma è veramente in esecuzione. L'illusione — per così dire — che più programmi siano eseguiti contemporaneamente viene ottenuta con una tecnica di *partizione del tempo* o *time sharing*: i diversi programmi in esecuzione sono in attesa e la CPU esegue una parte di ciascuno a rotazione; il kernel del sistema operativo si preoccupa di assegnare le risorse (in particolare unità centrale e memoria volatile) a ciascuno di essi a turno, in base anche ad una tabella di priorità. Se la potenza di calcolo è sufficientemente elevata, ogni utente ha l'impressione di avere l'intero sistema a sua disposizione. Si tratta evidentemente di una soluzione utilissima laddove sia necessaria una elevata possibilità di interazione fra utente e macchina, ma piuttosto onerosa per il sistema operativo che aumenta notevolmente la sua complessità.

Dal punto di vista dell'utente il calcolatore appare con una struttura "virtuale" che è diversa dalla macchina fisica e dipende non solo dalla costituzione dei suoi componenti (cioè dall'hardware) ma anche, e soprattutto, dalla struttura del sistema operativo. Ad esempio, se il sistema è multiprogrammato l'utente ignora che il suo programma è in competizione con altri per l'uso della CPU; oppure nei sistemi dotati di memoria virtuale è come se l'utente avesse a disposizione per il suo programma una memoria le cui dimensioni possono essere superiori a quelle fisiche della memoria principale. Considerazioni analoghe valgono per le periferiche. Un sistema multiutente appare quindi come un insieme di calcolatori virtuali, uno per ogni terminale, ciascuno dotato di una memoria virtuale, un insieme di archivi personali, uno per ogni utente, ed un archivio comune, contenente i programmi di utilità come ad esempio i compilatori di programmi per i varî linguaggi. Gli archivi personali sono accessibili solo agli utenti autorizzati tramite una *chiave* o *parola d'ordine* (*password*) che il sistema richiede prima di permetterne l'accesso; d'altra parte un qualsiasi archivio personale è accessibile da qualunque terminale, e sotto certe condizioni è possibile trasferire informazioni da un archivio personale ad un altro. Nelle applicazioni tecnico–scientifiche il sistema centrale appare all'esterno come un insieme di calcolatori e archivi disgiunti tra loro, poiché l'attività di ogni utente è in genere del tutto separata e indipendente da quella degli altri (fatte salve le possibilità di scambio di informazioni fra archivi di utenti diversi): in questo caso la centralizzazione del sistema si giustifica con la possibilità di concentrare le risorse in genere e la potenza di calcolo in particolare mettendole al servizio di quegli utenti collegati ad un certo istante e solo per il tempo richiesto. Invece nelle applicazioni organizzative e di controllo dei processi la centralizzazione è richiesta dalla necessità di coordinare l'attività di più utenti nell'ambito di una stessa procedura. In tutti i casi la struttura virtuale del sistema dipende dallo specifico sistema operativo installato.

3.2.1 Sistemi operativi a interfaccia grafica e a interfaccia di schermo

I numerosi sistemi operativi possono essere grossolanamente suddivisi in due grandi categorie. I sistemi operativi *a interfaccia grafica* vengono utilizzati e gestiti manipolando delle immagini grafiche sullo schermo che ne rappresentano i varî elementi. Ad esempio i files sono indicati oltre che da un nome anche da una piccola immagine, detta *icona*, diversa a seconda del tipo di file (di testo, di dati, eseguibile, e così via), e sono racchiusi all'interno di *finestre* che rappresentano le cartelle che li contengono. Tutta la gestione dei files, come la loro copia o spostamento fra cartelle o la loro rimozione, avviene mediante la selezione ed il trascinamento sullo schermo delle relative icone per mezzo del mouse; anche l'esecuzione dei programmi, che spesso creano altre finestre per comunicare con l'utente, è attivata e controllata col mouse. E' inoltre possibile la selezione di elementi e opzioni da menu a tendina di cui le varie finestre sono più o meno tutte dotate. Nei sistemi operativi *ad interfaccia di schermo* invece tutta la gestione avviene attraverso l'introduzione da tastiera di stringhe alfanumeriche. L'interfaccia con l'utente è anch'essa di tipo alfanumerico, i comandi, di sistema o di utente, vengono invocati col loro nome, e sia l'ingresso che l'uscita dei programmi in esecuzione sono ancora costituiti da caratteri alfanumerici.

Il sistema operativo Unix, di cui si tratterà d'ora in poi, è eminentemente ad interfaccia di schermo, sebbene le versioni più aggiornate contengano anche una interfaccia grafica con la quale è possibile eseguire la maggioranza delle operazioni. Tuttavia le interfacce grafiche su macchine diverse non sono standardizzate, e sarebbe difficile sceglierne una come esempio esplicativo. Al contrario i comandi di linea sono in gran parte gli stessi anche su modelli molto diversi, e pertanto è certamente più conveniente usare questi ultimi, dato che in questo modo ciò che viene imparato su una macchina può essere poi efficacemente applicato su molte altre. A tal fine occorre far notare che tutte le suddette interfacce grafiche contengono comunque uno o più emulatori di terminali alfanumerici.

§ 3.3 Il BIOS

Il *BIOS* (per *Basic Input Output System*) è un insieme di programmi di servizio che eseguono operazioni di base, di basso livello e di supervisione del calcolatore*. Il loro

* Il termine BIOS viene di solito adoperato nell'ambito dei personal computer, mentre in altri sistemi lo stesso gruppo di programmi aventi la medesima funzione è più spesso indicato con *firmware*.

compito è di prendersi cura di tutte le necessità dei componenti elettronici, dispensando i programmi di livello più alto dal conoscerne l'esatto funzionamento. Infatti i componenti specifici installati in un modello possono essere lievemente diversi da quelli di modelli simili per esempio meno recenti, avere funzionalità e modi di accesso o di risposta differenti (come indirizzi di memoria o porte logiche per l'ingresso o l'uscita), o persino contenere altre schede elettroniche; se non esistesse il BIOS, ogni programma dovrebbe conoscere i dettagli del loro funzionamento e non sarebbe facilmente trasportabile da una macchina all'altra. I programmi del BIOS fanno da interfaccia fra tali programmi e l'elettronica: i primi quando devono interagire con i componenti del calcolatore, non lo fanno direttamente ma chiamano delle funzioni del BIOS, le quali invece conoscono le specifiche del sistema e come interrogarlo, e sono esse stesse poi a passare i risultati ai programmi chiamanti. Per questo il BIOS è generalmente conservato in un chip ROM (o più spesso EEPROM) montato all'interno del calcolatore stesso.

I programmi contenuti nel BIOS si possono suddividere in due gruppi, quelli che vengono eseguiti una sola volta all'accensione del calcolatore e quelli che invece vengono interpellati durante tutto il suo funzionamento. All'accensione infatti il BIOS esegue innanzitutto una serie di autotest (detti per questo *POST* ovvero *Power-On Self-Test*) con le quali controlla lo stato e le condizioni dei principali componenti del calcolatore (dischi, memorie, video e altre periferiche). Terminati questi test prende avvio una procedura di inizializzazione, che prepara il calcolatore al suo uso, tra l'altro creando la tabella dei vettori di interrupt (si veda il par. 2.3.6) e impostando l'equipaggiamento della macchina: molte componenti infatti necessitano l'impostazione di registri e parametri per poter essere pronte per l'uso. Al tempo stesso il BIOS rileva pure il tipo di hardware installato interrogando le schede presenti sul bus.

Naturalmente è possibile che alcune componenti non possano essere gestite dai programmi standard del BIOS, ma abbiano bisogno di un supporto particolare. Durante il processo di avvio il BIOS riconosce tali componenti e per questo cede temporaneamente il controllo al loro BIOS specifico, posto su chip delle stesse schede: quest'ultimo può eseguire configurazioni particolari, come ad esempio aggiornare la tabella dei vettori di interrupt, in modo che all'occorrenza venga invocata non la funzione generica del BIOS di sistema bensì la funzione specifica del BIOS della scheda. Quando questo ha terminato il proprio lavoro di inizializzazione ripassa il controllo al BIOS principale che prosegue nel suo compito.

Al termine dell'inizializzazione il BIOS cerca una periferica (in genere un disco fisso) da cui caricare il vero e proprio sistema operativo, e cede ad esso il pieno controllo del calcolatore.

L'altro gruppo di programmi del BIOS è costituito dalle funzioni di gestione dei servizi e degli interrupt. Le prime si occupano di interrogare le periferiche per conto dei programmi di più alto livello, mentre le seconde, come si è visto nel par. 2.3.6, rispondono alle richieste di attenzione provenienti dall'elettronica stessa.

§ 3.4 Il caricamento del sistema operativo

Quando hanno termine i controlli e l'inizializzazione del sistema, il BIOS cerca una periferica da cui caricare il sistema operativo. Generalmente si tratta di un disco fisso, ma per effettuare l'avvio del sistema può essere sufficiente anche un dischetto, un CD-ROM o su alcuni sistemi pure una unità nastro. Questa operazione di caricamento nel suo insieme prende il nome di *bootstrap* o semplicemente *boot*. Ogni sistema ha un proprio metodo per fare ciò, per cui ci si limiterà a descrivere i passi logici essenziali del caricamento da disco fisso, senza scendere nei dettagli.

Innanzitutto il BIOS determina da quale partizione deve essere caricato il sistema operativo. Questa informazione può essere contenuta in una EEPROM, oppure lo stesso BIOS può esaminare la tabella delle partizioni dei dischi per individuarne una come "attiva" cioè contente il sistema (*bootable partition*); una variante è costituita dai personal computer, il cui BIOS non è in grado di fare ciò da solo, per cui carica ed esegue un breve programma contenuto nel primo settore del disco fisso, il cosiddetto *master boot record*, o *MBR*, quello stesso che contiene anche la tabella delle partizioni, ed è questo programma che svolge il compito di cercare una partizione attiva. Trovatane una, in un modo o nell'altro, il BIOS carica un altro piccolo programma, il *boot loader*, che è contenuto nel primo settore della partizione attiva, il *boot sector*, e gli cede il controllo; è il boot loader che effettivamente carica la parte più importante del sistema operativo, il suo kernel (o, nei sistemi più semplici, l'interprete di comandi). Questo passaggio intermedio ha due vantaggi: innanzitutto solleva il BIOS dal compito di conoscere "dove" esattamente si trova il kernel, cioè in quali blocchi del disco. Infatti il kernel è da questo punto di vista un file come gli altri, e la dislocazione di un file su disco dipende dal tipo di sistema di gestione dei files installato; i dettagli di quest'ultimo però sono noti solo al kernel, che ancora non è attivo; d'altra parte inserire queste informazioni nel BIOS lo renderebbe molto pesante, senza contare che difficilmente si riuscirebbe ad essere easustivi comprendendo tutti i possibili sistemi di gestione. La soluzione di gran lunga più semplice è appunto quella in cui il BIOS carica un programma da una locazione fissa, per esempio il primo settore, ed è poi compito di questo programma sapere dove si trova il kernel; se viene cambiato sistema di gestione dei files, è sufficiente aggiornare tale programma andando a modificare un solo settore del disco, operazione di gran lunga più semplice che non modificare il BIOS. Come secondo vantaggio, nel caso in cui siano presenti più d'un sistema operativo questa tecnica consente di scegliere uno di essi in modo semplice passando dei parametri a questo programma. Il boot loader infine carica in una posizione predefinita della memoria il kernel e passa a questo il controllo.

Il kernel inizia determinando una serie di informazioni essenziali sul calcolatore, come il tipo di processore installato, la quantità di memoria RAM presente, il numero e il tipo di dischi, il tipo di bus e le schede installate, in particolare la scheda video; può fare ciò per mezzo del BIOS oppure interrogando direttamente i varî componenti. In questo processo il kernel fissa una serie di parametri dipendenti dal modello delle periferiche i quali consentono ai sottosistemi di controllo di gestirle efficacemente. Dopodiché il kernel inizia a creare una serie di processi di sistema che rendono usabile il calcolatore. Quest'ultima fase è strettamente dipendente dal tipo di sistema operativo, e quindi ci si limiterà a descrivere lo Unix.

3.4.1 Il caricamento dello Unix

Il kernel di un sistema Unix, analogamente ad altri sistemi, come prima operazione determina il tipo di hardware disponibile sul calcolatore; per ogni componente crea una tabella con le informazioni necessarie ai moduli che effettivamente lo controlleranno. Poi crea un processo, che sarà per così dire il "padre" di tutti i processi, e quindi entra in uno stato di sospensione, pronto ad intervenire su richiesta di altri processi.

Il processo creato direttamente dal kernel esegue un programma particolare, chiamato `init`, che ha il compito di portare il sistema ad uno stato usabile. Per fare ciò lancia una serie di altri programmi secondo una successione prefissata che esso determina leggendo alcuni files di configurazione. L'ordine esatto varia da sistema a sistema, ma in genere tutti comprendono i seguenti passi:

- `init` determina il tipo di sistema di gestione dei files presente su ogni partizione di ciascun disco fisso; per ognuno di tali sistemi verifica che sia in buone condizioni, cioè che le informazioni contenute siano coerenti ed eventualmente tenta di correggerle (*file system check*); quindi li rende disponibili al sistema operativo;
- inizializza le periferiche presenti facendo uso delle informazioni raccolte dal kernel nonché di altri files di configurazione; se necessario fa partire dei programmi che ne consentono l'uso;
- assegna un nome al calcolatore; nel caso poi questo sia collegato ad una rete di comunicazioni, avvia una serie di servizi che permettono di utilizzarla (ad ogni servizio corrisponde un processo che esegue il programma adatto);
- fa partire dei programmi, che raccoglieranno informazioni sul funzionamento del calcolatore ad intervalli regolari o quando avvengono determinati fatti (come ad esempio registrare quando e per quanto tempo gli utenti ne fanno uso);
- infine crea uno o più processi che consentono agli utenti di collegarsi al calcolatore; questi processi possono attendere comandi provenienti direttamente dalla tastiera oppure restare in ascolto delle porte sulle quali sono connessi dei terminali; se disponibile, in genere viene avviata anche una interfaccia grafica.

A questo punto il calcolatore è pronto per l'uso. Il processo di `init` a sua volta resta in sospeso, in attesa di rientrare in azione al momento di spegnere il calcolatore.

3.4.2 Lo spegnimento del calcolatore

Dato il grande numero di processi in esecuzione ad ogni istante, non è possibile semplicemente spegnere il calcolatore quando questo non serve più: è necessario invece svolgere una sequenza di operazioni analoga al bootstrap ma in ordine inverso; questa sequenza prende nome di *shutdown*. Per fare ciò viene inviato un comando particolare ad `init`, il quale riprende il controllo ed esegue una serie di comandi che consentono di lasciare il sistema in uno stato affidabile. `init` invia un segnale a tutti i processi presenti affinché terminino correttamente il proprio lavoro (ad esempio salvando su disco i dati su cui stavano operando e chiudendo gli eventuali files aperti); quindi mette al sicuro i dati su disco scrivendo quelli eventualmente presenti solo nella memoria volatile; infine segnala al kernel di cessare le sue attività ed esce. Solo ora è possibile togliere l'alimentazione al calcolatore.

Fra tutti i componenti i dischi sono sicuramente i più delicati in tutto questo processo. Infatti per ottimizzare le prestazioni il kernel conserva parte delle informazioni nella memoria volatile, la cui velocità di accesso è molto maggiore, e solo periodicamente le scrive sui dischi. Quindi in caso di improvvisa mancanza di alimentazione i dati sul disco rimangono in uno stato incoerente. E' per questo motivo che il sistema li controlla attentamente prima di renderli disponibili, per evitare che leggendo o scrivendo in zone non disponibili venga compromessa l'integrità dei dati. Il programma che effettua tali controlli è di solito in grado di correggere i più comuni errori, ma seppur raramente a volte il danno è di tale entità da non permettere di utilizzare la partizione incriminata senza prima effettuarne una nuova formattazione, il che implica la perdita totale dei dati prima contenuti.

§ 3.5 Il sistema operativo Unix

Il sistema operativo Unix fu ideato nel 1969 da Ken Thompson, ricercatore presso i Bell Laboratories dell'American Telephone and Telegraph Company a Murray Hill nel New Jersey, utilizzando un calcolatore PDP-7. La prima versione fu scritta in assembly ed incorporava molte caratteristiche di MULTICS, uno dei primi sistemi operativi multiutente a partizione del tempo sviluppato presso il Massachussets Institute of Technology. Nel 1972 Thompson decise di riscrivere Unix in un linguaggio di alto livello affinché fosse trasportabile su macchine diverse, e per questo scelse un linguaggio detto B. L'anno successivo un altro ricercatore dei Bell Laboratories, Dennis Richie, modificò profondamente questo linguaggio per renderlo adatto alla progettazione e alla scrittura di sistemi operativi, e lo chiamò C. La terza versione di Unix fu appunto scritta in C, e ciò permise di adoperarlo su calcolatori più potenti, come il PDP-11/45 e il PDP-11/70 prodotti dalla Digital Equipment Corporation. Da allora lo Unix uscì dai laboratori e cominciò a diffondersi prima nelle università e quindi anche in ambienti industriali.

La possibilità di essere installato su molte piattaforme differenti costituisce certamente uno dei punti di forza dello Unix e ha contribuito notevolmente alla sua popolarità: esso infatti rappresenta un ambiente software relativamente stabile per un mercato hardware in rapido cambiamento. L'essere scritto in un linguaggio di alto livello invece che direttamente in linguaggio macchina offre ai produttori di calcolatori la possibilità di utilizzare un sistema operativo relativamente a basso costo per i loro nuovi prodotti, in quanto non è necessario sviluppare da zero un sistema *ad hoc*, ma basta procurarsi il codice sorgente e scrivere solo un compilatore per il C ed il codice di gestione delle periferiche installate. Per lo stesso motivo molte applicazioni possono essere direttamente portate da una piattaforma ad un'altra a costi molto bassi, spesso semplicemente ricompilando il codice sorgente.

Questa caratteristica dello Unix però ne rappresenta anche per alcuni versi un punto debole. Infatti i varî produttori tendono a "personalizzare" la versione di Unix installata nei proprî calcolatori, ad esempio aggiungendo nuove opzioni ai comandi o modificandone il comportamento o la locazione, al fine di renderlo preferibile ai modelli della concorrenza, e questo anche a seguito dell'allentamento dei diritti d'autore e delle licenze sul codice sorgente da parte della AT&T . Esistono quindi differenti versioni di Unix, tutte molto simili fra loro e tuttavia con caratteristiche concettuali a volte profondamente diverse, ma le due principali sono la *BSD* (*Berkeley Software Distribution*), sviluppata presso la Berkeley University a partire dal 1974, da quando cioè lo Unix iniziò a diffondersi nelle università americane, e la *System V*, introdotta sin dal 1983 dalla stessa AT&T. Tutte le altre versioni si rifanno ad una di queste due: a puro titolo di esempio si citano AIX (Advanced Interactive Executive) installato su macchine IBM, HP-UX su macchine Hewlett–Packard, SunOS e Solaris su macchine Sun Microsystems, ULTRIX (oramai in disuso) e OSF/1 (Open Software Foundation) su macchine Digital, e non va dimenticato Linux, un sistema operativo tipo Unix ma riscritto da zero in grado di operare su molte architetture diverse compresi i personal computer dotati di processori Intel. Da parecchi anni esiste pure uno standard Unix non proprietario, detto *POSIX*, definito dall'IEEE, ma finora pochi produttori vi aderiscono pienamente, molti preferendo ancora introdurre nelle proprie versioni alcuni elementi distintivi non standard.

Va detto per altro che alcuni prodotti sviluppati inizialmente da singole compagnie sono poi diventate di uso comune su tutte le piattaforme. Rilevante è il caso di NFS, NIS e X-windows, i primi due sviluppati dalla Sun Microsystems, il terzo dal Project Athena, un consorzio fra IBM, Digital e MIT. *NFS* (*Network File System*) consente un accesso trasparente ai dischi installati su altri calcolatori in rete come se fossero collegati alla macchina locale; *NIS* (*Network Information Service*) permette di centralizzare e quindi ridistribuire ai calcolatori in rete informazioni utili quali nomi e gruppi di utenti o parole chiave di accesso; *X-windows* fornisce uno standard per le interfacce grafiche e i sistemi a finestre per macchine Unix di diverso tipo.

§ 3.6 La gestione della memoria

Uno dei compiti del kernel di un sistema operativo è la gestione della memoria. Tale gestione è certamente uno degli aspetti più complessi, ancorché più interessanti per capirne il funzionamento, e pertanto ci si limiterà ad esporre i concetti fondamentali sulla sua implementazione.

Durante l'esecuzione di un programma il processore legge una istruzione dalla memoria, la decodifica e la esegue, quindi passa all'istruzione successiva; nell'eseguirla può essere necessario leggere o scrivere un dato in memoria. Pertanto il processore accedde continuamente alle celle della memoria o per scaricare una istruzione oppure per caricare o scaricare un dato. La situazione più semplice è quella in cui ad ogni istante l'intero programma è presente nella memoria fisica: in questo caso il processore fa riferimento direttamente agli *indirizzi fisici* (*physical addresses*) delle celle per operare lettura e scrittura. Questa soluzione però può andar bene nei sistemi uniprogrammati, nei quali un solo programma alla volta è in esecuzione, ma non è certo applicabile ai sistemi multiprogrammati, innanzitutto perché difficilmente tutti i programmi da eseguire potrebbero trovar spazio contemporaneamente nella memoria fisica (anzi, potrebbe anche accadere persino nei sistemi uniprogrammati che un singolo programma sia talmente grande da non poter essere contenuto tutto in memoria, quantunque dato il basso costo e l'alta capacità delle memorie RAM odierne è questa una opportunità assai poco probabile), e in secondo luogo perché sarebbe assai difficile se non impossibile mantenere separati i varî programmi affinché non si intralcino fra loro, soprattutto se ognuno contiene solo indirizzi assoluti.

La soluzione comune è l'uso della cosiddetta *memoria virtuale* (*virtual memory*). In questo caso il programma fa riferimento non agli indirizzi fisici della memoria, bensì ad *indirizzi virtuali* (*virtual addresses*), i quali vengono convertiti in indirizzi fisici dal processore in base a tabelle mantenute dal kernel. Per facilitare questa conversione l'insieme degli indirizzi virtuali viene suddiviso in porzioni della stessa dimensione, per esempio 4 Kb, dette *virtual pages* (*pagine virtuali*); anche l'insieme degli indirizzi fisici viene suddiviso in porzioni di dimensione pari a quella delle pagine, dette *page frames*. Ad entrambe viene assegnato un numero progressivo, detto *virtual page number* per le prime e *page frame number* per le seconde. Il processore converte un virtual page number in un page frame number per mezzo di tabelle, dette per questo *page tables*.

Quando deve essere eseguito un programma, il sistema lo cerca sul disco ma ne carica in memoria soltanto la prima pagina virtuale, sistemandola in una page frame, quindi stabilisce una corrispondenza fra indirizzi virtuali e indirizzi fisici. Si supponga ad esempio che la pagina 0, contenente gli indirizzi virtuali da 0 a 4095, sia stata sistemata nella page frame 5, contenente gli indirizzi fisici da 20480 a 24575; se il processore deve eseguire l'istruzione contenuta all'indirizzo virtuale 132, prima lo converte nell'indirizzo fisico corrispondente, pari a $20480 + 132 = 20612$, e poi va a leggere la corrispondente cella della memoria fisica. Se al contempo deve eseguire un altro programma, ne caricherà la prima pagina virtuale in un'altra page frame libera e ne inizierà l'esecuzione allo stesso modo. Ne segue che è necessaria una page table per ogni programma in corso.

Finché gli indirizzi virtuali si trovano all'interno della stessa pagina virtuale, il processore può tranquillamente proseguire nell'esecuzione del programma. Prima o poi però una qualche istruzione farà riferimento ad un indirizzo presente in un'altra pagina virtuale; nell'esempio precedente potrebbe essere necessario leggere un dato contenuto nell'indirizzo

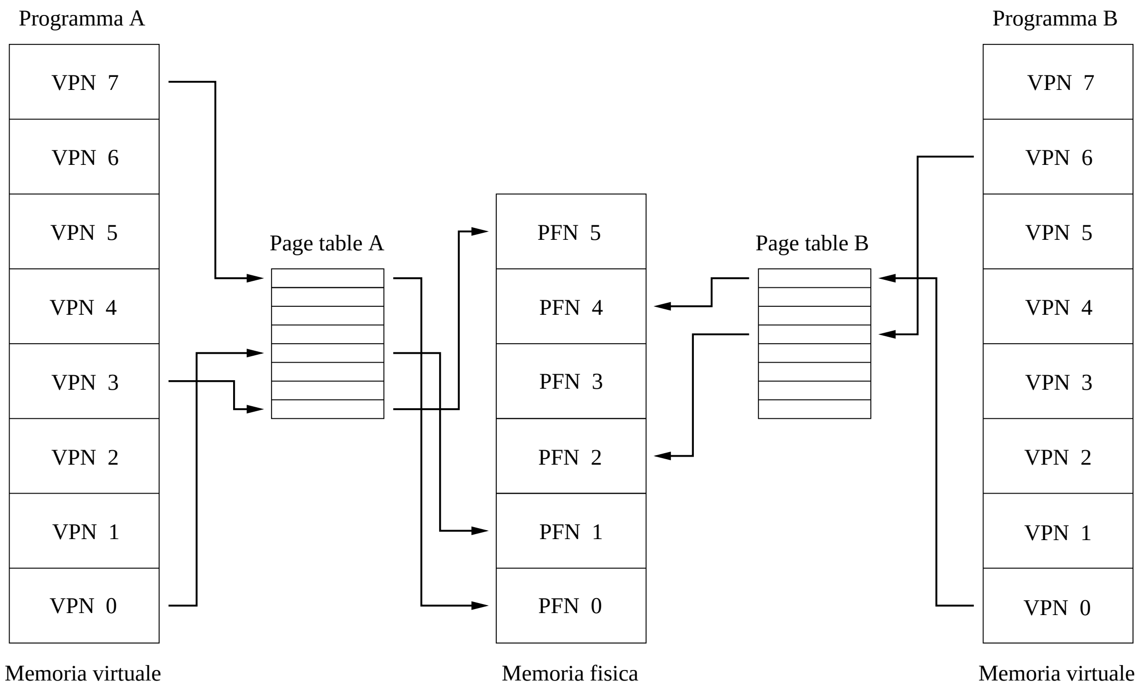

Fig. 3.5 *Due programmi in esecuzione, ciascuno con alcune pagine virtuali correntemente nella memoria fisica (VPN = Virtual Page Number, PFN = Page Frame Number)*

virtuale 10100. In questo caso il processore non sa come tradurlo in indirizzo fisico, e allora genera un segnale particolare, detto *page fault*. In presenza di tale segnale il kernel interviene andando a cercare sul disco la pagina virtuale in cui si trova l'indirizzo richiesto, nell'esempio la numero 2 che contiene gli indirizzi virtuali da 8192 a 12287, e caricandola in una page frame libera, ad esempio la numero 9; al termine la page table corrispondente viene aggiornata. Ora il processore sa dove trovare nella memoria fisica l'indirizzo che gli serviva, e può continuare ad eseguire il programma. Questo metodo di operare, per cui le pagine vengono caricate solo quando servono, è noto come *demand paging*. Poiché però la ricerca e la lettura dal disco di una pagina richiede un tempo molto lungo nella scala del processore, mentre il primo programma è sospeso in attesa della sua nuova pagina il processore può eseguire altri programmi che erano a loro volta in attesa: tutto questo, come già detto altre volte, rende il processore più efficiente e simula l'esecuzione in parallelo di più programmi. In Fig. 3.5 è esemplificata la situazione di due programmi in esecuzione.

Ad un certo punto tuttavia può accadere che non ci siano più page frames libere in cui sistemare una pagina virtuale richiesta. In questo caso il kernel deve fare spazio scartandone alcune (in realtà il kernel interviene già prima, quando il numero di page frame libere scende al di sotto di una soglia minima, ma il concetto è lo stesso). Esistono svariati metodi per determinare quali frames eliminare, ma il più diffuso sceglie quelle usate meno di recente (e per questo detto *LRU* per *Least Recently Used*), perché è meno probabile che queste possano ancora servire in futuro. Se la page frame da scartare contiene del codice, può essere semplicemente cancellata, giacché se dovesse servire ancora può essere riletta dal disco. Se invece contiene dei dati, deve essere prima salvata da qualche parte per poter essere poi recuperata. A tal fine viene riservata una zona del disco (generalmente, ma non necessariamente, una partizione apposta), detta *swap area* o *area di scambio*: in essa il kernel salva le pagine contenenti informazioni che altrimenti andrebbero perse. Quando poi il processore tenterà di accedere ad uno dei loro indirizzi, genererà nuovamente un segnale di page fault, e il kernel si occuperà di recuperare la pagina necessaria o dall'area di scambio o dal programma su disco. E' importante che l'algoritmo di scelta delle page frames da scartare sia efficiente: se infatti vengono frequentemente cancellate pagine che poi serviranno a breve, il kernel sarà costretto a rileggere di continuo le pagine dal disco, il che rallenta notevolmente i programmi in esecuzione, sempre fermi ad attendere le pagine necessarie.

L'uso della memoria virtuale presenta molti vantaggi. Innanzitutto il sistema appare come se avesse molta più memoria di quanta ne abbia in realtà: infatti la memoria virtuale può essere molte volte maggiore della memoria fisica. Questa tecnica consente poi a più programmi di essere eseguiti contemporaneamente, avendo ognuno a disposizione tutta la memoria che gli occorre e senza che si intralcino fra loro, giacché ciascuno ha il proprio spazio di indirizzi virtuali cui il sistema fa corrispondere una porzione della memoria fisica; per di più è il kernel che effettua materialmente le operazioni di lettura e scrittura della memoria a beneficio dei programmi in esecuzione, il che impedisce ad un programma non solo di modificare le aree di pertinenza di altri programmi, ma anche di danneggiare per errore le proprie, per esempio scrivendo dati in una zona contenente del codice. Al tempo stesso questo metodo consente a due o più programmi di scambiarsi informazioni leggendo e scrivendo in un'area comune della memoria, sempre però sotto il controllo del kernel. Infine questo meccanismo permette al sistema di assegnare equamente ai varî programmi una risorsa preziosa come la memoria fisica e ciò in modo assolutamente trasparente: ogni programma infatti "ignora" di essere in competizione con altri e che solo una parte di sé è presente in memoria, venendo eseguito come se fosse l'unico e con a disposizione una memoria grande quanto gli occorre.

Tutti i programmi vengono eseguiti facendo uso della memoria virtuale, tranne uno: il kernel stesso viene sempre eseguito nello spazio degli indirizzi fisici, e ciò è ovvio dovendo esso gestire il meccanismo stesso della memoria virtuale. A tal fine una parte della memoria fisica viene permanentemente riservata a contenere il kernel e non può essere utilizzata da altri; né il kernel può mai essere spostato nell'area di scambio, giacché esso stesso contiene il codice per effettuare tale operazione e quella inversa di recupero di una pagina dalla swap. Per questo motivo il codice del kernel contiene indirizzi assoluti della memoria fisica e non indirizzi relativi di una memoria virtuale.

§ 3.7 I processi

Un programma è un insieme di istruzioni e dati registrati su un supporto, ad esempio un disco, e come tale è una entità passiva e statica. Un processo è un "programma in azione". Si tratta quindi di una entità attiva e dinamica, che cambia continuamente mano a mano che il codice viene eseguito. Un processo infatti comprende non solo il programma in esecuzione, ma anche i registri della CPU, in particolare il program counter, i dati e i risultati parziali delle elaborazioni in corso e le zone di memoria in cui essi sono immagazzinati, nonché le informazioni sui files in uso: in poche parole tutta l'attività corrente del processore.

Lo Unix è un sistema operativo multiprogrammato, in cui cioè molti processi sono attivi in un qualunque istante. Ogni processo è un soggetto a sé con i suoi privilegi e le sue responsabilità; ognuno è isolato per evitare che interferisca negativamente con gli altri (se per esempio un processo si interrompe bruscamente per un errore di programmazione, non deve per questo provocare l'interruzione di altri processi), e se deve comunicare con un altro lo fa solo attraverso meccanismi affidabili gestiti dal kernel. Ciascun processo fa uso di molte risorse di sistema per portare a termine il suo compito: utilizza la CPU per eseguire le istruzioni del programma, la memoria RAM per contenere tali istruzioni e i dati in corso di elaborazione, i dischi e altri supporti per leggervi o scrivervi le informazioni che gli necessitano, e così via. Il kernel deve tenere traccia di ogni processo e delle risorse a lui assegnate, al fine di gestirle equamente fra tutti ed evitare che alcuni le monopolizzino a scapito degli altri.

La risorsa più preziosa è la CPU. Quasi sempre vi sono più processi in esecuzione che processori accessibili — tipicamente uno solo —: la maggior parte dei processi quindi è sospeso in attesa che un (o il) processore sia libero. Quando un processo in esecuzione deve attendere che un'altra risorsa sia disponibile, per esempio un dato da leggere sul disco, il processore invece di aspettare senza far nulla (come avverrebbe in un sistema uniprogrammato) sospende il processo in corso e ne prende in esecuzione un altro; quando la risorsa sarà disponibile, ad esempio quando il dato sarà stato rintracciato sul disco e copiato in memoria, il primo processo potrà riprendere. Poiché operazioni come lettura e scrittura dal disco richiedono un tempo relativamente lungo, nel frattempo la CPU può eseguire molte istruzioni di altri programmi. Pertanto ogniqualvolta un processo deve attendere il completamento di una operazione, il kernel lo sospende e assegna la CPU ad uno dei processi in attesa. Può accadere tuttavia che il processo in corso non abbia bisogno di risorse per un lungo tempo: per evitare che gli altri processi debbano aspettare troppo, il sistema controlla da quanto tempo esso è in esecuzione, e quando supera un intervallo prefissato (detto *time slice*) viene comunque sospeso e la CPU assegnata ad un altro.

La gestione di un meccanismo così complesso è affidata ad una parte del kernel, lo *scheduler*. Essa usa un algoritmo per scegliere quale sia il processo successivo da eseguire: a tal fine ad ognuno viene assegnato un numero intero, detto *priorità*, in base al quale tale scelta viene operata: processi con priorità più alta vengono ripresi più di frequente, e quindi in media restano in esecuzione per un tempo maggiore.

Un'altra risorsa preziosa è la memoria RAM. Per far sì che ogni processo abbia spazio sufficiente per eseguire il proprio programma, il kernel ne carica solo una parte, quella correntemente in uso, lasciando il resto sul disco e cancellando o riversando nell'area di scambio le parti al momento non utilizzate, come illustrato precedentemente discutendo della memoria virtuale.

Fra i molti dati che il kernel conserva per ogni processo vi sono lo stato in cui si trova (in esecuzione, sospeso, e così via), il suo identificatore (un numero intero che serve al kernel per identificarlo univocamente), la sua priorità, l'utente e il gruppo di appartenenza, i files ad esso collegati, la directory di partenza e quella attuale, e alcune informazioni sulla sua memoria virtuale.

In Unix tutti i processi, tranne il primo, hanno un *processo genitore* (*parent process*): ogni nuovo processo non è creato dal nulla, ma piuttosto è copiato, o per meglio dire "clonato", da un altro. Durante la fase di caricamento del sistema operativo, quando il kernel termina l'inizializzazione crea il primo processo, il quale esegue un programma particolare detto `init`; quest'ultimo genera tutti gli altri processi presenti nel sistema. Per esempio fra gli altri crea un processo che esegue un programma, `getty`, il quale attende una richiesta di accesso da parte di un utente. Quando questi inserisce il proprio identificatore, `getty` crea un altro processo che esegue il programma `login`, il quale verifica se l'utente sia autorizzato all'accesso (di solito richiedendo una parola chiave); in caso positivo `login` a sua volta genera un altro processo che esegue un interprete di comandi, deputato appunto a ricevere comandi dall'utente. Quando egli ne invia uno, l'interprete crea un nuovo processo in cui lo esegue.

Per creare un processo figlio il processo genitore chiama una funzione di sistema (*fork*) che ne duplica tutte le strutture, registri, codice di programma, dati, eccetera; una seconda funzione di sistema (*exec*) rimpiazza il programma che il processo genitore stava eseguendo con quello che il processo figlio dovrà eseguire; quindi il kernel crea una nuova struttura di dati per accogliere le informazioni relative al nuovo processo. Infine lo inserisce nella coda dei processi attivi: quando arriverà il suo turno inizierà l'esecuzione del suo codice.

§ 3.8 Il sistema di gestione dei files

Uno dei punti di forza dello Unix è l'organizzazione del suo sistema di gestione dei files, o *file system*. Contrariamente ad altri sistemi operativi (come DOS e Windows ad esempio), in Unix non si accede ai varî dischi e alle partizioni di ciascun disco mediante un identificatore di unità diverso per ciascuno, come un numero, una lettera o un nome, bensì sono tutti combinati a formare una singola struttura gerarchica la quale rappresenta l'intero sistema di gestione come un'unica entità. Ogni periferica appare come una sottodirectory di un unico albero, e l'utente non deve preoccuparsi di dove fisicamente i files si trovino: può trattarsi di una partizione sullo stesso disco come di un disco differente e persino di uno situato su un'altra macchina, appaiono tutti come se fossero un solo disco locale. Quando il kernel deve rendere disponibile una nuova area, la *monta* agganciandola ad una directory (che prende nome di *mount point*): da quel momento il contenuto precedente della directory diventa invisibile (per questo solitamente si usano directory vuote come mount point) mentre appare il contenuto dell'area montata. Se il kernel li contempla, è possibile montare sistemi di gestione diversi fra loro e tutti si comporteranno come se fossero parte di un unico disco. Infatti il sistema operativo considera ogni periferica come una sequenza lineare di blocchi: è compito dello specifico gestore della periferica in questione (*disk driver*) tradurre la richiesta di lettura o scrittura di un dato blocco nei termini che sono significativi per la geometria di tale periferica, cioè tracce, settori e cilindri. Con questo metodo ogni sistema di gestione dei files si comporta allo stesso modo nei confronti del sistema operativo, qualunque sia la periferica fisica che lo contiene.

Il sistema di gestione dei files in Unix presenta una directory principale, la *directory radice* o *root directory*, da cui tutte le altre dipendono; molte di queste hanno nomi e funzioni standard su macchine diverse. Questa non è nient'altro che la directory radice di una partizione particolare, detta per questo *periferica radice* (*root device*), che contiene i programmi basilari del sistema operativo e dalla quale viene anche caricato il kernel. Tuttavia, come spiegato nel primo paragrafo, ogni partizione che contenga un sistema di files ha una sua directory radice (ed eventualmente delle sottodirectory): è proprio questa directory che viene associata al mount point quando il sistema monta detta partizione.

I files sono identificati dal *percorso*, o *path*, che conduce a loro a partire dalla directory radice; tale percorso contiene la lista delle directory attraversate, i cui nomi sono separati da barre: per esempio

$$\texttt{/usr/src/linux/Documentation/devices.txt} \tag{3.1}$$

identifica il file `devices.txt` contenuto nella directory `Documentation`, che sta nella direc-

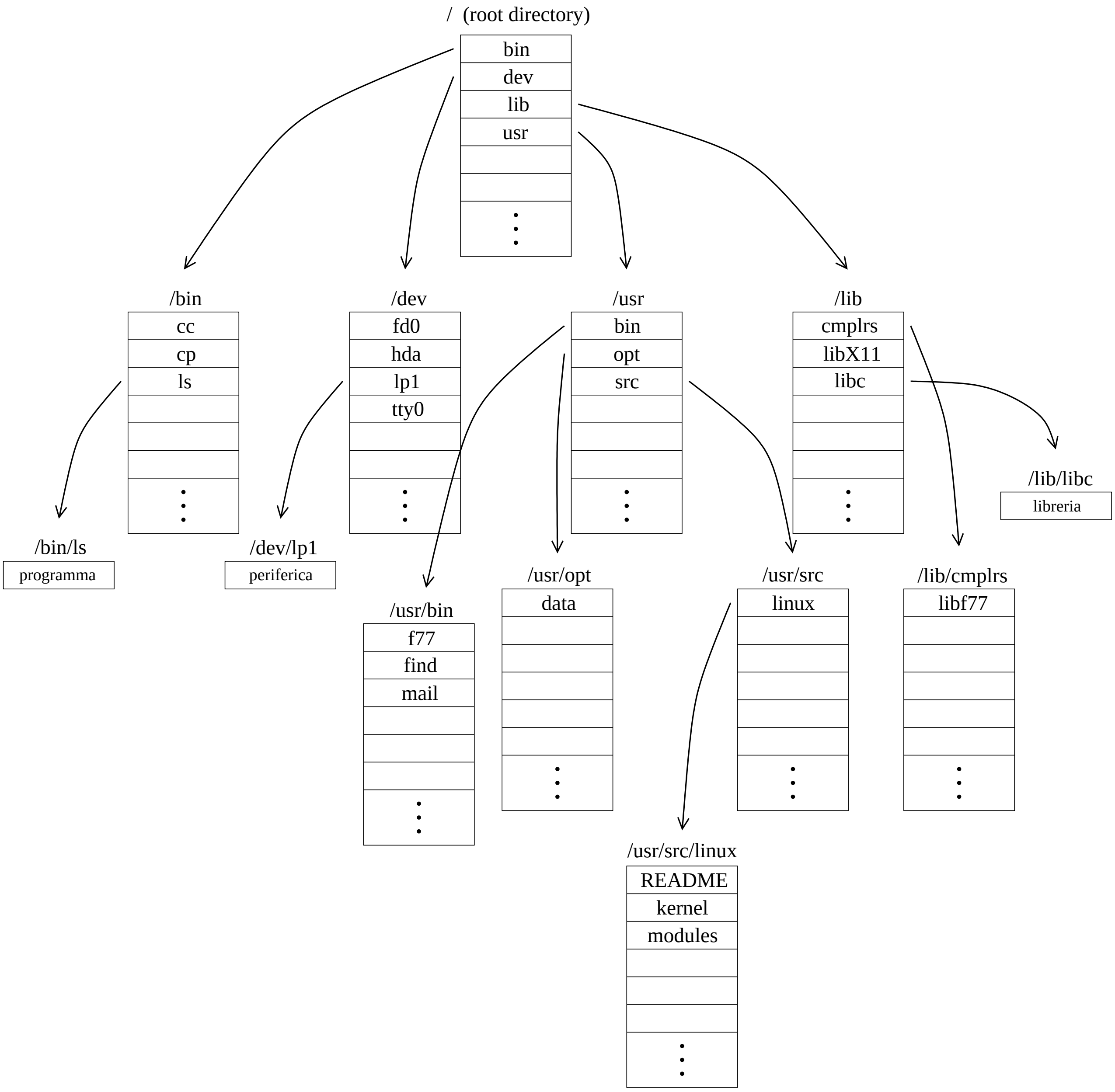

Fig. 3.6 *Schema di una parte di una tipica struttura di directory in Unix*

tory `linux`, la quale è una sottodirectory di `src`, che a sua volta è contenuta nella directory `usr`, la quale è una sottodirectory della directory radice, indicata dalla prima barra `/`. In Fig. 3.6 è rappresentata schematicamente una parte dell'albero delle directory di un sistema Unix. Da notare che lo Unix distingue le lettere maiuscole dalle minuscole, per cui ad esempio i nomi `dati`, `Dati` e `DATI` indicano tre files diversi. Ad ogni istante ciascun processo ha una *directory di lavoro* (*working directory*); il percorso di un file può essere assoluto, cioè riferito a partire dalla directory radice, oppure relativo, cioè riferito alla directory di lavoro corrente.

Associata ad ogni file vi è una mappa di bit che indica gli accessi consentiti. Tale mappa contiene tre campi di tre bit l'uno: il primo campo si riferisce ai permessi accordati al proprietario del file (*user*), il secondo quelli accordati al gruppo cui il file appartiene (*group*), e il terzo a tutti gli altri utenti (*others*). Dei tre bit il primo controlla la possibilità che il file venga letto (R per *Read*), il secondo che venga modificato (W per *Write*), e il terzo che venga eseguito (X per *eXecute*)*: per esempio se un file ha protezioni `RWX R-X --X` può essere letto, modificato ed eseguito dal suo proprietario, letto ed eseguito dagli utenti del gruppo, e solo eseguito da tutti gli altri (ovviamente si tratta di un programma eseguibile, altrimenti il bit di execute non avrebbe significato). Anche le directory hanno una analoga mappa di nove bit per regolarne gli accessi, in cui R indica la possibilità di leggere i nomi dei files in essa contenuti, W la possibilità di modificare tali nomi, mentre la X consente di leggere gli attributi dei files. Per poter cancellare un file occorre avere sia l'accesso in scrittura alla directory che l'accesso in scrittura al file.

Di ciascun file viene conservato non solo il nome dell'utente che ne è il proprietario, ma anche il nome del suo gruppo. Gli utenti infatti possono essere suddivisi in gruppi, e ogni utente può appartenere a più di uno. In questo modo è possibile garantire accessi diversi agli utenti che fanno parte di un dato gruppo e agli altri che non ne fanno parte (per esempio, tutti gli utenti potrebbero essere in grado di eseguire un certo programma, ma solo coloro che appartengono al gruppo di quelli che lo sviluppano potrebbero anche leggerne il contenuto).

In Unix tutte le informazioni su un file sono contenute in un blocco particolare, detto *i–node*. Al momento della formattazione di un disco o di una partizione un numero predeterminato di blocchi viene riservato quasi all'inizio per contenere non i dati ma le informazioni sui files; questi blocchi speciali, numerati progressivamente per poter essere facilmente identificati, costituiscono appunto gli i–node. Ogni file ha bisogno di un i–node che lo descriva, e quindi di fatto il numero massimo di files che un disco o una partizione può contenere è fissato una volta per tutte. Ciascun i–node contiene tutte le informazioni rilevanti di un file tranne il nome, che è invece conservato nella directory: questa infatti è semplicemente un elenco di nomi di files ognuno associato ad un i–node. Quando il sistema

* Esistono altre protezioni, ma meno usate, queste sono le principali.

deve leggere i dati di un file, ne cerca il nome nella directory e ricava il numero di i–node in cui i suoi dati sono registrati. Pertanto quando viene cambiato il nome di un file nulla di esso viene modificato se non la corrispondente voce nella directory. Da questo punto di vista poi anche le directory sono considerate files come gli altri, quindi anch'esse hanno ciascuna un i–node che le descrive, solo che il loro contenuto è particolare: contengono infatti elenchi di altri files. Così se il sistema deve leggere il contenuto del file dell'esempio (3.1), cerca il nome `usr` nella directory radice, ricava da essa il numero del suo i–node, e usa questo per leggerne il contenuto; verifica che `usr` è in effetti una directory, cerca al suo interno la voce `src`, ne estrae il numero di i–node e ne scorre il contenuto alla ricerca di `linux`; e così di seguito fino a determinare l'i–node corrispondente al file richiesto.

Le informazioni contenute in un i–node sono il tipo di file cui esso si riferisce (per esempio se si tratta di un file regolare oppure di una directory), la mappa di bit che descrive gli accessi, il nome del proprietario ed il gruppo, la dimensione del file in byte, la data e l'ora in cui l'i–node è stato creato, in cui è stato letto per l'ultima volta e in cui è stato modificato per l'ultima volta. Seguono poi 15 indirizzi di blocchi su disco che consentono di localizzare i dati contenuti nel file in questione. I primi 12 indirizzi puntano ad altrettanti blocchi che contengono i dati del file (questi vengono detti *blocchi diretti* o *direct blocks*): in questo modo si possono descrivere tutti i files lunghi non più di 12 blocchi (pari a 12288 byte nell'ipotesi di blocchi da 1 kB l'uno). Per files più grandi il tredicesimo indirizzo punta ad un blocco (detto *blocco indiretto* o *indirect block*) che contiene gli indirizzi di 256 blocchi di dati: in questa maniera si descrivono i files non più lunghi di $12 + 256 = 268$ blocchi (274432 byte sempre nell'ipotesi di blocchi da 1 kB). Per files ancora più lunghi il quattordicesimo indirizzo nell'i–node punta ad un blocco (*blocco doppiamente indiretto* o *doubly indirect block*) che a sua volta contiene gli indirizzi di 256 blocchi indiretti. Si può così arrivare a $268 + 256 \times 256 = 65804$ blocchi (67 383 296 byte). Se ciò non fosse sufficiente, il quindicesimo indirizzo punta ad un blocco (*blocco triplamente indiretto* o *triply indirect block*) il quale contiene gli indirizzi sul disco di 256 blocchi doppiamente indiretti: si arriva in questo modo ad un massimo di $65804 + 256 \times 256 \times 256 = 16843020$ blocchi (17 247 252 480 byte, quasi 17 GB)*. In Fig. 3.7 è mostrato lo schema di un

* In alcuni dischi più piccoli ogni blocco di tipo indiretto contiene solo 128 indirizzi di blocchi di livello

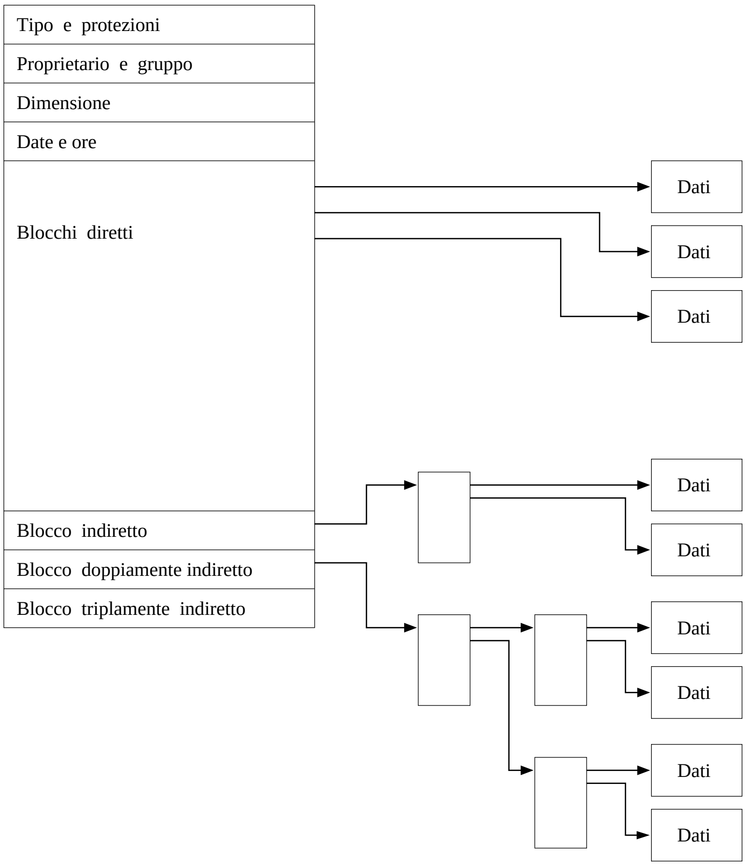

Fig. 3.7 *Schema di un i–node*

i–node.

Da quanto detto si deduce che un file è identificato dal suo i–node, non dal nome nella directory. Quindi è possibile che più nomi, nella stessa directory o in altre, si riferiscano allo stesso file. Esiste per questo una chiamata di sistema per creare un *collegamento*, o *link*, ad un file già esistente, ovvero per inserire in una directory una voce che si riferisce ad un file già presente su disco: in questo modo esso appare in due directory senza però dover duplicare lo spazio occupato. E' ovviamente possibile anche creare una vera e propria copia del file in questione, ma in questo caso il file copiato avrà un suo proprio i–node e occuperà il suo spazio su disco. Quando si rimuove un file in realtà si elimina solo il suo collegamento nella directory; allorché il numero di collegamenti scende a zero, l'i–node viene rilasciato e lo spazio occupato dal file marcato come libero.

Accanto ai files ordinarî lo Unix prevede anche una seria di *files speciali* usati per accedere alle periferiche di ingresso e uscita. Quest'ultimo aspetto mostra la grande poten-

inferiore invece di 256, ma la filosofia è la stessa.

zialità del sistema di gestione dei files in Unix: presentando le componenti dell'hardware come files viene facilitata enormemente la gestione delle periferiche, in quanto lo scambio di dati con esse si riduce per l'utente all'apertura, lettura, scrittura e chiusura di un file speciale, queste operazioni venendo svolte mediante le usuali chiamate di sistema usate pure per i files regolari. I files speciali, solitamente raccolti nella sottodirectory `/dev`, non occupano spazio su disco, ma sono solo un punto di accesso al gestore della periferica cui si riferiscono. Questa caratteristica è registrata opportunamente nei rispettivi i–node. Quando il sistema cerca di leggere o scrivere in uno di questi files, il kernel riconosce il loro carattere speciale e interpella il gestore della rispettiva periferica. Vi sono infatti files speciali per i dischi, sia IDE che SCSI, i terminali, le porte seriali e parallele, i nastri magnetici, le schede di rete, le schede audio, e molto altro.

§ 3.9 La shell

Quando un utente accede al sistema, questi crea per lui un processo in cui esegue un programma particolare, l'*interprete di comandi* o *shell*: tale programma legge dal terminale i comandi inseriti dall'utente e invia le opportune chiamate di sistema al kernel per eseguirli a suo beneficio. Ci sono essenzialmente tre tipi di interpreti, la *Bourne shell* (`sh`), la *Korn shell* (`ksh`) e la *C shell* (`csh`); essi differiscono per la sintassi di alcuni comandi e per altre caratteristiche peculiari (soprattutto nella scrittura di *procedure*, files contenenti una serie di istruzioni da eseguire), ma i principî di base sono gli stessi. Esistono comunque versioni diverse di ciascuna shell, più o meno evolute: trattandosi infatti di programmi utente ognuno è libero di scriversene una e di usarla come proprio interprete.

Come si è detto, lo scopo di una shell è di eseguire i comandi introdotti dall'utente. Un piccolo numero di comandi è inserito internamente nel codice della shell e vengono eseguiti direttamente da questa. La stragrande maggioranza invece è costituita da programmi di sistema. Quando l'utente invia un comando la shell cerca un programma con lo stesso nome; se lo trova crea un sottoprocesso (cioè un processo figlio) che ha il compito di eseguirlo. Per tutto il tempo in cui il sottoprocesso è impegnato, esso ha il pieno controllo del terminale mentre la shell è sospesa; quando ha terminato il suo compito, il kernel distrugge il sottoprocesso e il controllo ritorna alla shell, che si pone in attesa del comando successivo.

La shell ha un elenco di directory in cui cercare i programmi da eseguire, elenco che l'utente può modificare e integrare. La shell cerca il programma voluto in ciascuna directory nell'ordine in cui appaiono nell'elenco, e appena ne trova uno si ferma e lo esegue, senza verificare se nelle directory successive ve ne sia un'altra copia. L'utente comunque può sempre inserire il comando corredato dal percorso completo per forzare la shell ad eseguire proprio quella versione.

Normalmente la shell rimane sospesa fintantoché il comando inserito non ha termine. Tuttavia se un programma impiega molto tempo (e se non ha bisogno di interagire con l'utente), è possibile richiedere alla shell di ritornare immediatamente dopo aver creato il sottoprocesso: quest'ultimo rimane sempre un processo figlio della shell, ma perde il controllo del terminale che viene restituito subito all'interprete. L'utente comunque può sempre comunicare con i processi figli per mezzo della shell per sospenderli, esaminarli, farli ripartire o fermarli.

La shell inoltre consente all'utente di fornire i dati in ingresso a un programma da un file invece che dalla tastiera, come pure di dirigerne l'uscita verso un file invece che verso il terminale, oppure anche verso un altro comando: in questo modo è possibile concatenare più programmi uno dietro l'altro in modo che l'ingresso del successivo sia l'uscita del precedente. Per ciascun programma in questa catena (o *pipe*) la shell crea un sottoprocesso dedicato. Da questo punto di vista lo Unix permette una grande flessibilità, in quanto i programmi possono essere scritti in modo da leggere l'ingresso da e scrivere l'uscita verso unità generiche (dette *standard input* e *standard output*) che solo al momento dell'esecuzione vengono assegnate al terminale ovvero ad un file, ad un altro processo o ad un file speciale che descrive una periferica.

Bibliografia

A. S. Tanenbaum, *Structured Computer Organization*, ed. Prentice–Hall, 1984

D. A. Rusling, *The Linux Kernel* (reperibile in formato DVI e PostScript presso ftp://ftp.digital.com/pub/DEC/Linux–Alpha/Miniloader/docs/TheLinuxKernel e in formato HTML presso http://www.linuxdoc.org/LDP/tlk/tlk.html)

Altre informazioni sul kernel Linux possono essere reperite anche in http://jungla.dit.upm.es/˜ jmseyas/linux/kernel/hackers–doc.html

A UNIX saleslady, Lenore,
Enjoys work, but she likes the beach more.
She found a good way
To combine work and play:
She sells C shells by the seashore.

"The Linux User manual"

Capitolo IV

I programmi

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning," the King said, very gravely,
"and go on until you come to the end: then stop."

Lewis Carroll, "Alice's Adventures in Wonderland"

§ 4.1 Linguaggi di programmazione

Un programma è un insieme di istruzioni che il processore deve eseguire. Queste istruzioni però devono essere scritte nell'unico linguaggio che la CPU è in grado di capire, il linguaggio macchina. In Fig. 4.1 è mostrato come esempio una parte di un programma in linguaggio macchina e la sua traduzione in mnemonici.

I programmi scritti in linguaggio macchina sono immediatamente eseguibili dal processore, ma sono molto complicati da scrivere per un programmatore: questi infatti deve prendersi cura di una serie di dettagli assai noiosi, come ad esempio conoscere esattamente la locazione di memoria in cui un dato va inserito oppure l'indirizzo esplicito in cui si trova una certa istruzione da eseguire, senza contare poi che di ogni istruzione occorre ricordare il valore esadecimale. Per alleggerire almeno in parte questo lavoro abbastanza presto nella storia dei calcolatori fu inventato un linguaggio simbolico, l'assembly, che permette di usare sigle mnemoniche al posto dei valori numerici delle istruzioni e altri simboli e abbreviazioni per indicare le varie locazioni di memoria. Un programma scritto in assembly non è eseguibile dalla CPU, deve essere prima processato da un programma speciale, detto *assemblatore* (*assembler*), che traduca gli mnemonici nelle corrispondenti istruzioni e risolva le variabili simboliche nei valori effettivi.

```
0F79B    2E  DB  F7    ROL  $F7DB
0F79E    18            CLC
0F79F    AD  DA  F7    LDA  $F7DA
0F7A2    65  FE        ADC  $FE
0F7A4    85  FE        STA  $FE
0F7A6    AD  DB  F7    LDA  $F7DB
0F7A9    65  FF        ADC  $FF
0F7AB    85  FF        STA  $FF
0F7AD    A9  80        LDA  #$80
0F7AF    AE  DC  F7    LDX  $F7DC
0F7B2    F0  04        BEQ  $F7B8
0F7B4    4A            LSR
0F7B5    CA            DEX
0F7B6    D0  FC        BNE  $F7B4
0F7B8    A0  00        LDY  #$00
0F7BA    11  FE        ORA  ($FE),Y
0F7BC    91  FE        STA  ($FE),Y
0F7BE    60            RTS
```

Fig. 4.1 *Parte di un programma in linguaggio macchina*

Quantunque l'uso dell'assembly semplifichi in parte il compito dell'operatore umano, programmare in linguaggio macchina rimane un lavoro faticoso e facilmente suscettibile di errori di programmazione. Per questo oggi resta confinato ad alcune applicazioni molto specialistiche, spesso fortemente dipendenti dall'hardware in uso.

Col tempo sono stati sviluppati una serie di altri linguaggi di programmazione, detti *di alto livello* per distinguerli dall'assembly e dal linguaggio macchina. Questi linguaggi fanno uso di istruzioni simboliche, ognuna delle quali spesso richiede molte istruzioni in linguaggio macchina per essere portata a termine; tali istruzioni simboliche sollevano il programmatore dai compiti più noiosi come conoscere gli indirizzi di memoria, e gli permettono di concentrarsi su aspetti più elevati come la logica di sviluppo di un algoritmo. Oggi vi sono decine di linguaggi di alto livello, alcuni riservati ad ambiti specialistici, altri caduti in disuso, e di nuovi ne vengono sviluppati. A solo titolo di esempio si citano in ordine sparso FORTRAN, COBOL, BASIC, C, C++, Pascal, ADA, Perl, Java, SIMULA, Algol, PL/C, LISP, Tcl; anche gli interpreti di comandi di cui si è parlato alla fine del Capitolo precedente possono essere considerati linguaggi di programmazione.

I linguaggi di alto livello sono più vicini al modo corrente di affrontare e risolvere un problema quanto sono lontani dal modo di operare della macchina. Nessuno di questi linguaggi può essere eseguito dal processore così com'è, tutti vanno prima "tradotti" in linguaggio macchina. Il compito di traduzione è svolto dagli interpreti e dai compilatori.

4.1.1 Linguaggi interpretati e linguaggi compilati

I linguaggi di programmazione si suddividono in *linguaggi interpretati* e *linguaggi compilati*.

Per eseguire un programma scritto in un linguaggio interpretato si fa uso di un programma ausiliare, detto appunto *interprete* (*interpreter*), che lo traduce "al volo" e lo esegue immediatamente: ogni istruzione del programma viene letta, tradotta in linguaggio macchina ed eseguita, per poi passare all'istruzione successiva. Esempi di linguaggi interpretati sono Perl, LISP e su alcune macchine pure il BASIC, nonché gli interpreti di comandi, dove è la stessa shell che fa appunto da interprete. Il vantaggio dei linguaggi interpretati è che possono essere subito eseguiti senza passaggi intermedi, il che facilita il loro sviluppo e la ricerca di errori; lo svantaggio, a parte la necessità di avere sempre a disposizione l'interprete per poterli eseguire, è dato dal fatto che il programma deve essere tradotto nuovamente ad ogni esecuzione, con conseguenti perdite di tempo. E ciò vale non solo per l'intero programma: se si ha un ciclo ripetuto di istruzioni, ciascuna istruzione viene ritradotta ad ogni iterazione; così ad esempio se un gruppo di istruzioni deve essere ripetuto 100 volte, ciascuna di esse viene tradotta 100 volte prima di essere eseguita.

Per eseguire invece un programma scritto in un linguaggio compilato si fa uso di un altro tipo di programma ausiliario, detto *compilatore* (*compiler*), che lo traduce tutto in linguaggio macchina. Nessuna esecuzione ha luogo ora: il compilatore invece crea un secondo file in linguaggio macchina che è eseguibile dal processore in un momento successivo. Quasi tutti i linguaggi sono compilati, in particolare FORTRAN, C e C++. Vantaggi e svantaggi sono speculari rispetto ai linguaggi interpretati. Lo svantaggio è che ora è necessario un passaggio intermedio e la creazione di un secondo file; se si scopre e corregge un errore, tutta la procedura di compilazione va ripetuta. D'altra parte il programma viene tradotto una volta per tutte, e la versione in linguaggio macchina può essere eseguita infinite volte senza ulteriori traduzioni e anche se il compilatore non è più disponibile. Se un gruppo di istruzioni deve essere ripetuto 100 volte, ciascuna di esse viene comunque tradotta una volta sola, giacché solo nella fase successiva di esecuzione del programma il ciclo verrà effettivamente eseguito, e a quel punto nessuna traduzione ulteriore sarà più necessaria. Il file in linguaggio macchina prende talora anche il nome di *file binario* o *forma binaria*.

Dagli inizi degli anni '90 è diventato sempre più importante un terzo tipo di linguaggio ibrido che fa uso sia della compilazione che della interpretazione. Un programma scritto in uno di questi linguaggi viene tradotto da un compilatore in una forma binaria compatta, che però non è linguaggio macchina ma un cosiddetto *pseudocodice* (*pseudocode* o semplicemente *p-code*); al momento di eseguirlo questo pseudocodice viene interpretato. Lo pseudocodice è solitamente più semplice ma più potente del linguaggio macchina; la sua velocità di esecuzione è quasi paragonabile a quella di un linguaggio compilato, giacché gli interpreti sono piccoli e veloci, ma conserva la flessibilità e la potenza di un buon linguaggio interpretato. Esempi sono Python e Java.

§ 4.2 Compilazione di un programma

Il compito del compilatore è quello di trasformare il programma scritto in un linguaggio di alto livello (detto *programma sorgente*) nella corrispondente versione in linguaggio macchina (*programma eseguibile*). Se tutto il programma è contenuto in un unico file la compilazione può avvenire così come è stata descritta con un solo passaggio. Ciò tuttavia accade assai di rado: quasi sempre il programma da compilare è sparso su più files, e questo essenzialmente per due ragioni. La prima è che dal punto di vista organizzativo è generalmente più comodo suddividere il codice del programma in files diversi, contenenti ognuno una o più unità funzionali (*sottoprogrammi* o *funzioni*) magari affidate ad altrettanti sviluppatori. La seconda e più importante ragione è che spesso il programmatore vuole poter riutilizzare del codice già esistente senza doverlo riscrivere da capo: per esempio se un programma deve aprire un file o scrivere una riga sullo schermo non dovrebbe essere necessario riscrivere il codice che materialmente svolge queste operazioni, ma si vorrebbe poter usare delle funzioni di sistema già pronte. In effetti gli autori di un sistema operativo mettono a disposizione delle "librerie" di funzioni in grado di portare a termine operazioni legate al sistema (come appunto gestire un file o utilizzare lo schermo);

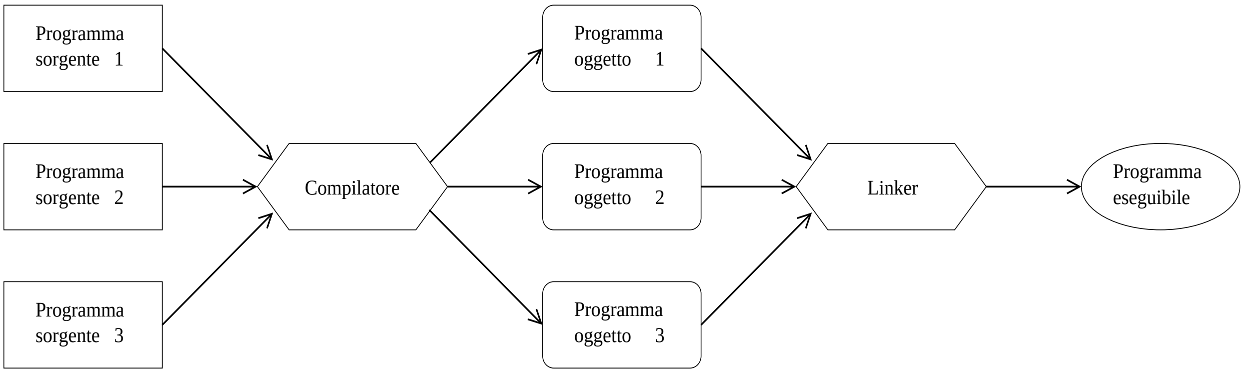

Fig. 4.2 *Esempio di compilazione di un programma costituito da tre moduli*

analogamente altri programmatori potrebbero sviluppare dei sottoprogrammi di utilità e metterli a disposizione come librerie. Una *libreria* infatti è una collezione di funzioni che possono essere incluse in un programma.

La compilazione avviene allora in due passi. Prima il compilatore vero e proprio trasforma ogni modulo in un programma intermedio, detto *programma oggetto*, che è sì scritto in linguaggio macchina, ma non è eseguibile perché mancano i riferimenti esterni alle altre unità funzionali. Interviene poi un secondo programma, detto *linker* o *link editor*, che raccoglie tutti i programmi oggetto e li fonde in un programma realmente eseguibile.

Si consideri l'esempio di Fig. 4.2 : tre moduli contenenti altrettante sezioni di programma sorgente vengono trasformate dal compilatore in tre files oggetto e quindi fusi dal linker in un programma eseguibile. Il primo vantaggio descritto in precedenza si concretizza nel fatto che se una funzione viene modificata non è necessario ricompilare tutto il programma, operazione dispendiosa in termini di tempo, ma solo il modulo interessato; ovviamente anche il processo di link va ripetuto, ma questo passaggio è molto più rapido, e quindi il risultato netto è un notevole risparmio di tempo. Senza contare come detto la flessibilità nello sviluppo di un grande programma, che può essere suddiviso in piccole sezioni più facilmente gestibili.

In Fig. 4.3 è mostrato un esempio analogo, in cui però il programmatore materialmente scrive solo una parte del codice (due moduli) mentre il resto è prelevato da una libreria al momento del link, il che come detto consente di inserire in generale sottopro-

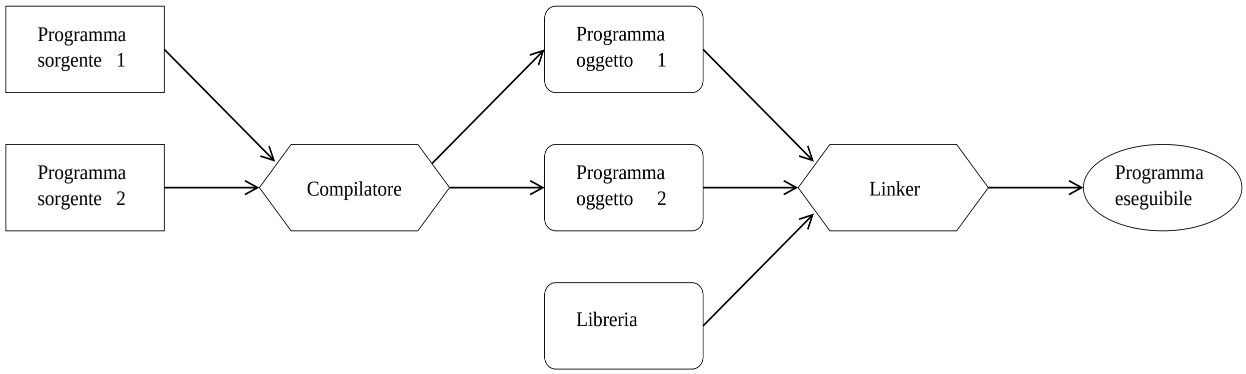

Fig. 4.3 *Esempio di compilazione di un programma costituito da due moduli ed una libreria*

grammi scritti da altri e in particolare funzioni e chiamate di sistema.

Quando il compilatore traduce un modulo, assume per semplicità che il codice corrispondente in linguaggio macchina inizi all'indirizzo virtuale 0; d'altra parte non sapendo a priori con quali altri moduli e in che posizione dovrà lavorare, deve necessariamente assumere un indirizzo di inizio arbitrario. In Fig. 4.4 sono mostrati in modo molto schematico i tre programmi oggetto immediatamente dopo la compilazione. Se essi fossero semplicemente caricati uno dietro l'altro in memoria, non potrebbero comunque essere eseguiti, dato che gli indirizzi in essi contenuti non sono coerenti, come appare in modo schematico in Fig. 4.5a. Si consideri ad esempio il modulo B: esso inizia con un salto all'istruzione contenuta nell'indirizzo (virtuale) 300, ma ora all'indirizzo 300 c'è il modulo A. Lo stesso modulo A in realtà non può iniziare da 0, giacché all'inizio di un programma, negli indirizzi più bassi, si trovano solitamente istruzioni di sistema che servono ad identificare e caricare il resto del programma (vettori di interrupt, funzioni di comunicazione col sistema operativo, e simili). Questo problema, noto come *relocation problem*, si manifesta perché ogni modulo oggetto rappresenta uno spazio di indirizzi virtuali separato. Il primo compito del linker è quindi di riaggiustare tutti gli indirizzi all'interno di ciascun modulo affinché siano coerenti con un'unica sequenza. Per fare ciò esso si costruisce una tabella con tutti i moduli e la loro lunghezza, e in base a questa assegna ad ognuno un indirizzo di caricamento (*load address*); quindi aggiunge ad ogni indirizzo in essi contenuto una costante (*relocation constant*) pari all'indirizzo di inizio di ciascun modulo.

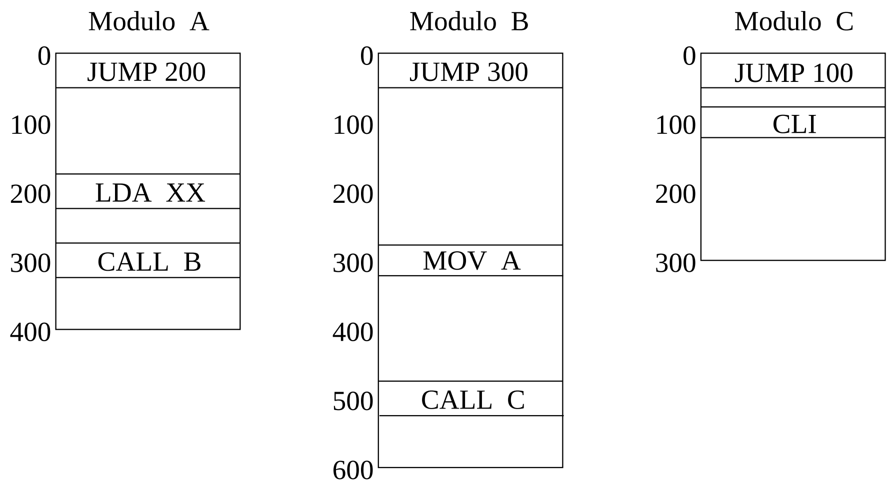

Fig. 4.4 *Schema degli indirizzi virtuali di tre moduli dopo la compilazione*

Un altro problema che il linker deve affrontare è noto come *external reference* ed è dovuto al seguente fatto: quando un modulo deve richiamare un altro modulo, deve usare un indirizzo simbolico, giacché non sa dove nel programma finale esso si troverà effettivamente. Il linker deve risolvere questi indirizzi simbolici introducendovi i valori effettivi. Nell'esempio il modulo A richiama il modulo B come sottoprogramma; ma il compilatore non può sapere dove si troverà tale modulo nel file eseguibile, per cui vi lascia un indirizzo simbolico; spetta al linker sostituirvi poi il valore reale, dato che a quel punto ogni modulo avrà trovato la sua sistemazione definitiva. La versione finale ora eseguibile (*absolute load module*) è mostrata schematicamente in Fig. 4.5b.

Quando l'operazione di link comprende delle librerie di sottoprogrammi, il programma eseguibile ottenuto può essere compilato in modo *statico* (*statically linked*) oppure in modo *dinamico* (*dinamically linked*). Nel primo caso il codice dei sottoprogrammi viene estratto dai files delle librerie e incluso nell'eseguibile. Nel secondo caso invece il linker si limita a introdurre nel file binario l'indicazione del punto d'inizio nella libreria dei sottoprogrammi richiesti; al momento dell'esecuzione quando un certo sottoprogramma sarà necessario verrà recuperato dalla libreria e caricato in memoria. Un eseguibile statico è pertanto più grande, giacché contiene pure il codice di tutti i sottoprogrammi di libreria, compresi quelli usati di rado o solo sotto particolari condizioni, ma proprio per questo può essere eseguito anche quando i files delle librerie non fossero più disponibili, ad esempio su macchine diverse da quella su cui è stato compilato. Al contrario un eseguibile dinamico è più piccolo,

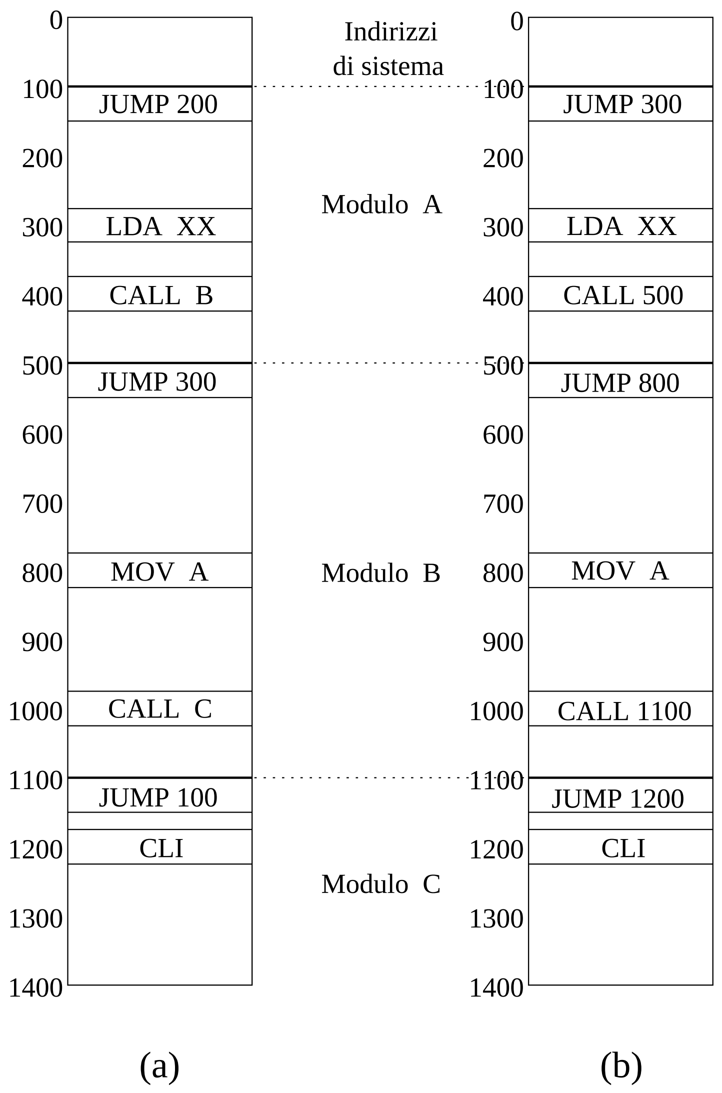

Fig. 4.5 *I tre moduli oggetto di Fig. 4.4 prima (a) e dopo (b) l'operazione di link*

visto che non contiene in sé i sottoprogrammi di libreria, e al momento dell'esecuzione vengono caricati in memoria solo quelli effettivamente necessarî; tuttavia è indispensabile la presenza dei files delle librerie, senza i quali l'esecuzione non può aver luogo. Se non viene esplicitamente richiesto il contrario, in genere i sottoprogrammi di sistema (cioè tipo quelli per aprire un file o leggere la tastiera) vengono inclusi in modo dinamico, poiché si suppone che le librerie di sistema siano presenti su tutte le macchine; altri tipi di librerie invece sono di solito inseriti in modo statico, a meno che l'utente non scelga il link dinamico.

Bibliografia

A. S. Tanenbaum, *Structured Computer Organization*, ed. Prentice–Hall, 1984

Mathematicians are like Frenchmen:
whenever you say something to them, they translate it into their own language
and at once it is something entirely different.

Johann Wolfgang von Goethe, “Maxims and Reflexions”

Capitolo V

Reti di calcolatori

Consider people living in the stone age, where drums may be used to transmit messages between individuals. Suppose caveman A wants to invite caveman B for a game of hurling rocks at each other, but they live too far apart for B to hear A banging his drum. So what are A's options ? He could 1) walk over B's place, 2) get a bigger drum, or 3) ask C, who lives halfway between them, to forward the message. The last is called networking.

Olaf Kirch, "The Linux Network Administrator's Guide"

Gran parte dell'importanza odierna dei calcolatori deriva dalla possibilità di comunicare fra loro per scambiarsi dati e informazioni, fino a costituire delle vere e proprie reti di calcolatori. In questo capitolo verranno fornite alcune semplici nozioni sul più diffuso protocollo di comunicazione e sui meccanismi usati per rendere possibili questi scambi. Data l'ampiezza dell'argomento, non si pretende in alcun modo di essere esaustivi: la progettazione, l'uso e la conduzione di reti di calcolatori infatti formano oggetto di interi Corsi a se stanti, la cui trattazione completa andrebbe ben oltre gli scopi delle presenti dispense. D'altra parte è relativamente semplice reperire documenti sul presente argomento sia in forma cartacea sia in forma elettronica effettuando delle ricerche.

§ 5.1 Introduzione

Una *rete* è un insieme di *nodi* in grado di comunicare tra loro; di solito un nodo è un calcolatore, ma può essere anche un terminale, una stampante o altro. Un *sito* è un gruppo di nodi logicamente collegati. Alcuni nodi dedicati permettono lo scambio di dati fra reti diverse.

La comunicazione fra calcolatori non sarebbe possibile senza un insieme di regole e un linguaggio comuni: un tale insieme prende nome di *protocollo*. Esistono molti protocolli di comunicazione, le cui regole sono stabilite di comune accordo oppure appartengono ad un marchio proprietario che le ha sviluppate; esistono così anche dei calcolatori specializzati in grado di "tradurre" un dato protocollo in un altro permettendo lo scambio di informazioni fra reti che utilizzano standard differenti.

La possibilità di collegare due o più calcolatori consente di realizzare moltissimi servizi di rete, che vanno dal trasferimento di interi file allo scambio di messaggi fra utenti, dalla condivisione di risorse (quali file e stampanti) alla esecuzione remota di programmi, e altri ancora.

Non tutti i protocolli sono così flessibili da consentire tutti questi servizi. Per esempio il protocollo *UUCP* (*Unix-to-Unix Copy*) permette un facile scambio di file fra una coppia di nodi (spesso, anche se non necessariamente, connessi da linea telefonica), eventualmente programmando l'orario del trasferimento in modo da non richiedere che i nodi siano perennemente collegati alla rete. Tuttavia i servizi descritti in precedenza, come la condivisione di risorse o l'esecuzione in interattivo di applicazioni remote, richiede un approccio diverso: invece di trasmettere interi file, i dati vengono suddivisi in piccole porzioni dette *pacchetti* (*packets*), che sono immediatamente inviati al nodo destinatario il quale successivamente si preoccupa di ricomporli. Un tale tipo di rete è detta *a commutazione di pacchetto* (*packet–switched network*).

Vi sono diversi protocolli a commutazione di pacchetto; il più diffuso è il TCP/IP.

§ 5.2 Il protocollo TCP/IP

Nel 1969 l'agenzia federale statunitense DARPA (*Defence Advanced Research Project Agency*) finanziò un progetto per la realizzazione di una rete di comunicazioni in grado di collegare in maniera affidabile i calcolatori della difesa militare. La rete sperimentale, denominata *ARPANET*, divenne operativa nel 1975. Nel 1983 l'insieme di protocolli TCP/IP fu adottato come nuovo standard di comunicazione, mentre ARPANET fu definitivamente abbandonata nel 1990. Nel frattempo anche enti governativi e università iniziarono a basare le proprie reti di calcolatori su questo protocollo, contribuendo alla sua diffusione mondiale.

A rigore sarebbe più corretto parlare di una "famiglia di protocolli Internet" di cui TCP e IP sono due membri. Tuttavia essendo questi i più noti, è entrato nell'uso comune chiamare "protocollo TCP/IP" l'intera famiglia di protocolli. Alcuni di questi servono a definire le informazioni che due nodi devono scambiarsi per portare a termine una operazione, come trasferire un file o inoltrare un messaggio di posta elettronica; altri invece forniscono servizi di più basso livello, come avviare a destinazione un pacchetto.

I protocolli Internet sono strutturati a livelli (*layers*): ogni livello fornisce dei servizi al livello superiore e sfrutta i servizi forniti dal livello inferiore. Questo consente ad ogni protocollo di preoccuparsi solo della gestione dei servizi che esso stesso deve garantire, e non dei dettagli dei servizi forniti da altri protocolli. In genere è possibile distinguere quattro livelli:

– al livello più alto c'è il protocollo di una singola applicazione, quale la posta elettronica o la condivisione di risorse
– vi sono poi protocolli, come TCP, che si preoccupano di suddividere i dati da trasmettere in blocchi, di controllare che arrivino a destinazione e di riassemblarli
– quindi vi sono protocolli, come IP, che si occupano di inviare i blocchi di dati a destinazione
– infine al livello più basso ci sono i protocolli che gestiscono l'hardware impiegato per la trasmissione dei dati

Il modello su cui TCP/IP si basa assume che esista un gran numero di reti indipendenti connesse fra loro: l'utente deve essere in grado di accedere alle risorse di una qualsiasi rete in modo completamente trasparente.

5.2.1 Il livello hardware. L'Ethernet

L'hardware più utilizzato nella realizzazione di reti locali (*Local Area Networks* o *LAN*) è l'*Ethernet*, essendo relativamente semplice ed economico da installare, gestire e mantenere.

Esistono tre tipi di Ethernet, *thick*, *thin* e *twisted pair*. Thick e thin Ethernet impiegano entrambi un cavo coassiale, di diverso diametro, per la trasmissione dei dati. Per collegare un nodo ad una rete thick Ethernet è necessario praticare un foro nel cavo e introdurvi un derivatore (*transceiver* o *vampire tap*); alla derivazione è poi possibile connettere una o più macchine: collegare nuovi nodi è quindi un'operazione un po' difficoltosa ma la rete non viene interrotta. Invece per allacciare un nodo ad una rete thin Ethernet occorre inserire un connettore a T di tipo BNC lungo il cavo e collegare l'estremo di tale connettore sul retro della macchina: questa operazione è più semplice ma la rete viene interrotta durante l'inserimento del connettore nel cavo. Thick Ethernet e thin Ethernet possono trasferire sino a 10 Mbit al secondo, e i cavi possono essere lunghi al massimo 500 e 200 metri rispettivamente: per questo vengono anche chiamati *10base-5* e *10base-2*. Una rete twisted pair Ethernet usa invece cavi costituiti da una coppia di fili di rame, molto simili agli ordinari cavi telefonici: collegare una nuova macchina è decisamente più semplice e non si verifica alcuna interruzione; tuttavia è necessario dell'hardware addizionale (*switch di rete*) per la distribuzione del segnale. Questo tipo di rete è noto anche come *10base-T* e *100base-T* a seconda che il trasferimento avvenga a 10 oppure a 100 Megabit al secondo; oggi però si trovano in commercio anche apparecchiature (switch e schede di rete) in grado di trasmettere fino a 1 Gigabit al secondo.

Il maggior inconveniente della tecnologia Ethernet è la limitata lunghezza dei cavi, che ne preclude l'uso al di fuori delle reti locali. Tuttavia è possibile collegare fra loro diversi segmenti Ethernet in modo che si comportino come se fossero un'unica rete: per questo si usano apparecchi come i *repeater*, che si limitano a copiare i segnali fra due o più segmenti, i *bridge* e i *router*, che invece analizzano i dati in ingresso e li inoltrano al segmento che contiene il nodo destinatario.

L'Ethernet trasmette *pacchetti* (*packets* o *frames*) grandi fino a 1500 bytes fra due nodi appartenenti alla stessa rete Ethernet. Il pacchetto trasmesso da una stazione viene visto da tutte le stazioni attaccate allo stesso segmento: ciò rappresenta una notevole semplificazione nella gestione della rete, in quanto non è necessario sapere a priori dove precisamente sia collegato il destinatario del pacchetto. Di norma ogni nodo semplicemente ignora i pacchetti non indirizzati ad esso, anche se è possibile barare ed esaminare il contenuto dei pacchetti indirizzati ad altri, il che rende potenzialmente insicure le trasmissioni Ethernet*. Se due stazioni cercano di trasmettere nello stesso momento, avviene una cosiddetta *collisione*, entrambe interrompono la trasmissione e riprovano alcuni istanti più tardi.

Per poter essere identificato ciascun nodo possiede un *indirizzo Ethernet* (*Ethernet address*) scritto in modo permanente nella scheda di rete (per questo sovente viene anche detto *hardware address*): per cambiare l'indirizzo Ethernet di una macchina è necessario sostituire fisicamente la sua interfaccia di rete. Un nodo può essere equipaggiato con più schede, connesse alla stessa rete Ethernet o a reti diverse. Questo indirizzo è costituito da 6 byte, solitamente scritti come una sequenza di numeri esadecimali a due cifre, ad esempio `08:00:15:A4:4B:59` . Ciascun fabbricante di apparecchiature Ethernet deve registrarsi presso una autorità centrale, per assicurare che gli indirizzi assegnati alla proprie periferiche siano diversi da quelli assegnati dagli altri costruttori: ciò garantisce che non vi siano due stazioni con il medesimo indirizzo Ethernet.

L'Ethernet è in grado di trasmettere sulla stessa rete dati con protocolli diversi, quali TCP/IP, DECnet, Xerox NS, eccetera.

5.2.2 Altri tipi di hardware

L'Ethernet comunque non è l'unico tipo di hardware utilizzato: altre tecnologie (e protocolli connessi) per le reti locali sono il *Token Ring* e l'*AppleTalk*.

Il Token Ring fu originariamente sviluppato dalla IBM negli anni '70, ed è tuttora secondo solo all'Ethernet in quanto a popolarità per le LAN. Tutte le macchine appartenenti alla rete sono collegate a formare un anello chiuso (da cui il nome), all'interno del quale viene fatto circolare un pacchetto particolare, detto *token*, il cui possesso garantisce il diritto di trasmettere. Se un nodo riceve il token e non ha nulla da trasmettere, semplicemente lo passa al nodo successivo (ogni stazione può trattenere il token per un tempo massimo definito). Se invece ha qualcosa da trasmettere, si impossessa del token, ne altera un bit particolare, vi aggiunge in coda gli indirizzi proprio e del nodo destinatario e i dati da trasmettere, e quindi lo passa al nodo successivo. Ogni nodo passa il pacchetto al vicino

---

* Gli switch moderni memorizzano l'indirizzo di ciascun nodo a cui sono collegati e vi inoltrano solo i pacchetti che gli competono, superando in gran parte questo problema.

finché esso non raggiunge il destinatario: questi si fa una copia dei dati e quindi continua a far circolare il pacchetto. Quando infine esso torna al nodo mittente, questi ne controlla l'integrità e che sia stato correttamente ricevuto dal destinatorio, quindi rilascia un nuovo token. Mentre il pacchetto sta circolando, nessun altro nodo è autorizzato a trasmettere, e pertanto in un Token Ring non si verificano mai delle collisioni. Inoltre questo tipo di rete, a differenza dell'Ethernet, è "deterministica", nel senso che può essere determinato a priori il tempo massimo necessario affinché un nodo sia in grado di trasmettere: questa caratteristica, insieme ad altre che ne garantiscono la robustezza, rendono il Token Ring ideale per le applicazioni dove conoscere il ritardo nella comunicazione è essenziale, per esempio nel campo dell'automazione industriale.

L'AppleTalk è un sistema di rete disponibile su tutti i calcolatori della Apple. Esso è in realtà un protocollo di trasmissione completo, che comprende sia le specifiche di basso livello per la trasmissione dei dati sui cavi sia le specifiche di alto livello per rendere disponibili servizi quali la condivisione di file e stampanti. Inizialmente il termine AppleTalk indicava sia il protocollo che lo standard dei cavi di trasmissione: poiché oggi esso è disponibile anche su macchine diverse con sistema operativo Unix e su supporti differenti, si preferisce usare i termini *LocalTalk* per i cavi originali Apple, *EtherTalk* quando AppleTalk viaggia su Ethernet, e *TokenTalk* quando impiega la tecnologia Token Ring.

Per reti su scala più grande, per esempio per connettere fra loro più LAN Ethernet, ci si serve spesso di cavi in fibra ottica, che impiegano il protocollo *FDDI* (*Fiber Distributed Data Interface*). Anche questo protocollo in sostanza prevede l'immissione in rete di un certo numero di token, ogni nodo rimanendo abilitato alla trasmissione di un pacchetto solo se ne "cattura" uno. Con questa tecnologia si può arrivare ad oltre 100 Mbit per secondo come velocità di trasmissione e a 200 km come lunghezza di cavo.

Per distanze ancora maggiori si adoperano tecnologie differenti, spesso basate sullo standard *X.25* .

Di frequente i radioamatori usano le proprie attrezzature per collegare i loro calcolatori: questa tecnica viene detta *packet radio* o *ham radio*, e in genere utilizza il protocollo *AX.25* (derivato dal precedente).

Infine non va dimenticata la possibilità di impiegare le normali linee telefoniche per la trasmissione dei dati. Diversi protocolli vengono usati in questo caso, come lo *SLIP* (*Serial Line Internet Protocol*), il *PLIP* (*Parallel Line Internet Protocol*) e il *PPP* (*Point-to-Point Protocol*).

5.2.3 Il protocollo IP

A questo punto far comunicare due calcolatori appartenenti alla stessa rete sembrerebbe semplice. Tuttavia non appare realistico poter connettere tutti i calcolatori esistenti con un'unica rete. D'altra parte si è visto come possano esistere diverse tecnologie di rete, ognuna con un proprio standard di trasmissione. Ovviamente si vorrebbe che le comunicazioni non fossero limitate ad una singola rete locale, ma si vorrebbe poter raggiungere e utilizzare una rete qualunque sia l'hardware impiegato per realizzarla e dovunque essa si trovi. Perché ciò sia possibile esistono dei nodi dedicati, detti *gateway*, che collegano fra loro due o più reti diverse: si tratta di macchine il cui compito è quello di scambiare pacchetti fra le reti alle quali sono collegate (mediante un'interfaccia differente per ciascuna). Per esempio, si supponga che un utente stia lavorando con la stazione `alpha` appartenente alla rete del Dipartimento di Scienze di una certa Università, e che egli intenda collegarsi alla stazione `beta` appartenente ad un diverso Dipartimento (Fig. 5.1). Poiché i calcolatori sono ubicati in edifici diversi, è assai probabile che facciano parte di due reti locali differenti: in questo caso non possono scambiarsi direttamente dei pacchetti Ethernet, ma devono far affidamento ai rispettivi gateway, il cui compito è appunto quello di connettere le due reti per mezzo di una terza rete in fibra ottica. Pertanto `alpha` spedisce i pacchetti al suo gateway `theta` tramite Ethernet; questi li inoltra tramite FDDI al gateway dell'altra rete `rho`, il quale li fa pervenire al destinatario finale `beta` ancora tramite Ethernet. Questo modo di scambiare dati fra due calcolatori remoti prende il nome di *routing* (letteralmente *instradamento*); in questo contesto i pacchetti spesso vengono denominati *datagrammi* (*datagrams*)*.

* Tecnicamente un datagramma è un insieme logico di dati che il protocollo tratta in maniera unitaria, mentre un pacchetto è un insieme fisico di segnali su un cavo (Ethernet o altro). In molti casi un pacchetto contiene semplicemente un datagramma, per cui i due termini sono spesso usati come sinonimi. Ci sono tuttavia delle eccezioni: per esempio, il protocollo X.25 suddivide un datagramma IP in pacchetti da

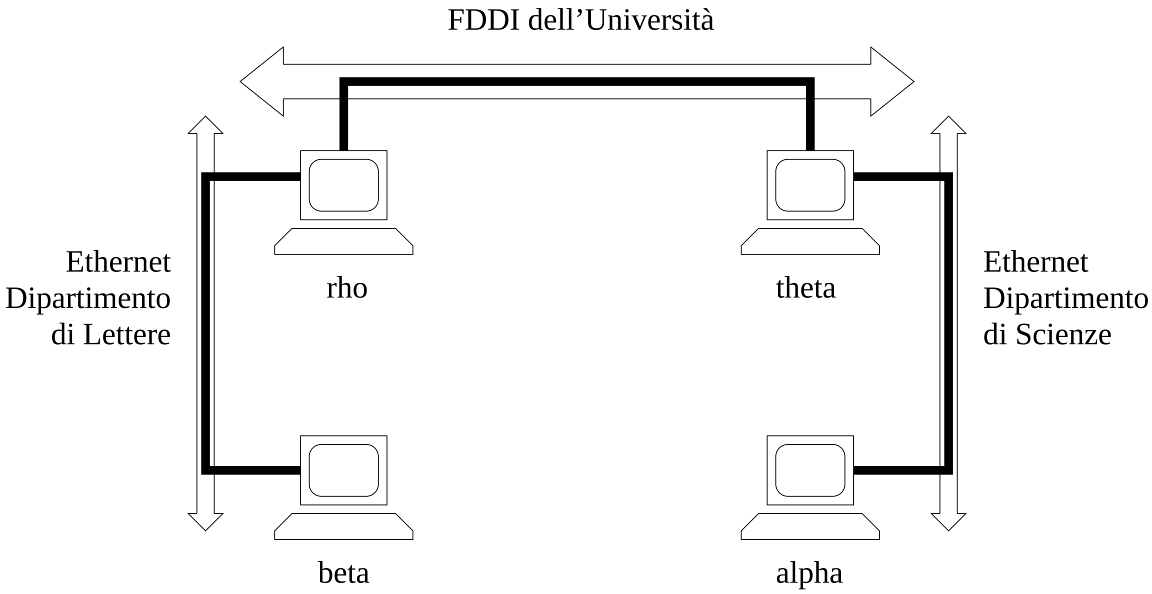

Fig. 5.1 *Scambio di un pacchetto fra due calcolatori appartenenti a due reti Ethernet diverse*

Per la gestione dello scambio di datagrammi appare evidente la necessità di un singolo protocollo di livello superiore che sia indipendente dall'hardware utilizzato: questo protocollo è l'*IP*, per *Internet Protocol*. Il vantaggio maggiore dell'IP è che esso è in grado di trasformare un insieme di reti fisicamente diverse in un'unica rete apparentemente omogenea: la "metarete" che ne risulta prende il nome di *internet*. Si noti la differenza fra una internet e Internet: la prima indica un qualsiasi insieme di reti diverse connesse dal protocollo IP, mentre il secondo è il nome ufficiale di una particolare internet globale.

Il compito del protocollo IP è semplicemente quello di trovare un possibile percorso per un datagramma dal nodo di partenza al nodo di destinazione e di inoltrarlo a quest'ultimo attraverso uno o più gateway. Nel fare ciò non deve preoccuparsi di quale tipo di hardware venga materialmente usato per la trasmissione, giacché questo è totalmente gestito dal sottostante protocollo di rete in maniera trasparente all'IP. Né deve interessarsi a cosa esattamente contenga il datagramma, essendo questo invece un compito del protocollo soprastante che sta utilizzando IP per la trasmissione. Un datagramma è un insieme di dati che viene spedito come un singolo messaggio, e ciascun datagramma è trattato indivi-

128 byte l'uno, per cui un singolo datagramma viene trasportato da più pacchetti. Questo processo è comunque trasparente all'IP, in quanto i pacchetti vengono riassemblati in un unico datagramma prima di essere passati al protocollo TCP/IP.

dualmente dalla rete, in maniera separata dai precedenti e dai successivi. Per esempio, si supponga di dover trasferire un file di 10000 byte fra due nodi: la maggior parte delle reti non può gestire pacchetti così grossi, per cui il protocollo superiore (per esempio TCP) scompone il file in più datagrammi, per esempio in 20 datagrammi da 500 byte l'uno. A questo punto IP trasmette ciascun datagramma da un nodo all'altro. Tuttavia mentre i pacchetti sono in viaggio, la rete non sa dell'esistenza di un collegamento fra di loro: è perfettamente possibile che un datagramma successivo, il decimo per dire, arrivi prima di un datagramma precedente, il nono ad esempio, oppure che fra due datagrammi del file in questione venga trasmesso il datagramma relativo ad un'altro file, o ancora che ad un certo punto della rete si verifichi un errore ed un certo datagramma vada ritrasmesso. Di tutto questo deve occuparsi il protocollo di livello più alto: compito dell'IP è unicamente quello di spedire il datagramma a destinazione.

Per poter operare con qualsiasi tipo di tecnologia di rete, l'IP adopera uno schema di indirizzamento indipendente dall'hardware. A questo scopo ad ogni nodo viene assegnato un *indirizzo IP* (*IP address*) univoco, del tutto indipendente dall'indirizzo hardware. A differenza di quest'ultimo, l'indirizzo IP è un elemento software e può essere cambiato utilizzando appositi programmi (alcuni sistemi operativi però richiedono il riavvio della macchina dopo il cambiamento dell'indirizzo IP). Questo indirizzo è un numero di 32 bit, solitamente scritto come quattro numeri decimali corrispondenti ai quattro byte, ad esempio `192.168.1.15` (questo formato viene detto *dotted quad notation*).

5.2.4 Il protocollo TCP

Naturalmente saper inviare datagrammi da un nodo ad un altro non assicura ancora la possibilità di comunicare: infatti i dati da trasmettere devono essere suddivisi in pacchetti dal nodo mittente e riassemblati correttamente dal nodo destinatario. Va poi notato che l'IP è stato intenzionalmente disegnato come un protocollo non affidabile: se il traffico generato è troppo grande, un gateway potrebbe momentaneamente trovarsi a corto di risorse e non essere in grado di inoltrare un datagramma. Quando questo accade, l'IP semplicemente lo scarta: il pacchetto è irrimediabilmente perso e va ritrasmesso. E' quindi compito dei nodi che stanno comunicando accertarsi della completezza e integrità dei dati trasmessi e ricevuti.

Tutto questo viene gestito da un protocollo di livello superiore, il *TCP* (per *Transmission Control Protocol*): sotto la sua responsabilità ogni messaggio viene suddiviso in pacchetti alla partenza e viene riassemblato all'arrivo, rimettendo ciascun pacchetto nell'ordine corretto e ritrasmettendo ciò che è andato perduto. Il TCP mette in opera una comunicazione affidabile facendo uso dei servizi forniti dall'IP, dando l'illusione che esista una connessione semplice, un "filo diretto" tra il nodo locale e la stazione remota, così da non doversi preoccupare di come e attraverso quale strada i dati effettivamente transitino: una connessione TCP è come un canale bidirezionale su cui entrambi i nodi possono leggere e scrivere.

Il protocollo TCP identifica i due estremi di una connessione per mezzo degli indirizzi IP dei due nodi coinvolti e del numero di una cosiddetta *porta* su ciascun nodo. Le porte sono come i punti di collegamento di una connessione e servono a tener traccia di differenti conversazioni. Ciascun servizio di rete ha un numero di porta standard che serve a distinguerlo, come sarà illustrato meglio nel seguito.

5.2.5 Altri protocolli

Il TCP è il protocollo più usato ma non certo l'unico a fare uso dell'IP.

Come si è visto, il protocollo TCP svolge molte operazioni, come scomporre e ricomporre i pacchetti e controllarne l'integrità. Tuttavia vi sono molti casi in cui tutta questa quantità di lavoro è inutile: per esempio quando i dati da trasmettere sono talmente pochi che sicuramente stanno in un unico datagramma, oppure quando non è necessario assicurarsi che il pacchetto sia arrivato perché questo controllo è già gestito dal protocollo di applicazione. In questi casi invece del TCP si usa il più semplice protocollo *UDP* (*User Datagram Protocol*): analogamente al TCP anche l'UDP consente ad una applicazione di contattare un servizio su una macchina remota, ma per fare ciò non stabilisce una connessione, semplicemente invia un singolo datagramma al destinatario. L'UDP non suddivide i dati in pacchetti, e non tiene traccia di ciò che è stato inviato per ritrasmetterlo in caso di errore. Anche l'UDP come il TCP usa gli indirizzi IP dei due nodi e i numeri di porta per inviare il pacchetto.

Uno dei software applicativi che usano UDP è NFS: questo protocollo permette di rendere disponibile un'area su disco fisicamente montata su una macchina server ad altre macchine in rete che si comportano da client. Quando un programma necessita di un file, manda una richiesta al server; tuttavia sarebbe troppo lungo stabilire una connessione TCP, inviare il file e quindi rilasciarla; invece il server invia al client il file suddiviso in alcuni pacchetti UDP, il che è assai più rapido. L'UDP però non è in grado di gestire i pacchetti persi o corrotti, compito che ricade quindi sull'applicazione, come nel caso di NFS.

Un altro protocollo alternativo al TCP è l'*ICMP* (*Internet Control Message Protocol*), che viene impiegato per comunicare messaggi di errore o informativi destinati non ad un particolare programma applicativo ma piuttosto allo stesso software di rete. Per esempio, tentando di collegarsi ad un nodo non raggiungibile la stazione di partenza riceve un messaggio ICMP di "Host Unreachable"; oppure se il TCP invia un pacchetto destinato ad una certa porta per la quale non vi è alcun processo sul nodo destinatario per gestirla, quest'ultimo invia un messaggio ICMP di "Port Unreachable" alla stazione di partenza. Un altro messaggio molto utile è l' "ICMP Redirect": si supponga che il software di rete del nodo `alpha` decida di spedire dei datagrammi al nodo remoto `beta` usando un certo percorso; se uno dei gateway attraverso cui i pacchetti devono passare si accorge, in base ai dati in suo possesso, che esiste un percorso più breve per andare da `alpha` a `beta` (dove più breve significa attraversando un minor numero di gateway), inoltra comunque i datagrammi lungo quel percorso, ma invia pure al nodo mittente un messaggio di "Redirect" segnalandogli la strada più breve, cosicché `alpha` la possa utilizzare per le successive comunicazioni con `beta`. L'ICMP è persino più semplice dell'UDP: non solo tutti i messaggi possono essere contenuti in un unico datagramma, ma non ha nemmeno bisogno di indicare un numero di porta, giacché tali messaggi vengono interpretati direttamente dal codice di rete stesso.

Esistono comunque anche dei protocolli che non usano IP. Uno di questi è l'AppleTalk, che, come si è già detto, comprende sia la gestione fisica di un modello di hardware sia la gestione logica di una rete di comunicazione. Ogni nodo di una rete AppleTalk possiede un indirizzo composto da due parti, un *numero di rete* (*network number*) e un *numero di* *nodo* (*node number*); quest'ultimo è compreso fra 1 e 253 e occupa un solo byte, mentre il primo va da 1 a 65535 e occupa due byte. Ad ogni indirizzo può essere associato un nome descrittivo per facilitarne l'uso da parte dell'utente. Vi sono poi dei router AppleTalk che permettono la comunicazione fra reti diverse. Un altro esempio di software di rete non basato sull'IP è il *DECnet* di proprietà della Digital. Si tratta in realtà di una famiglia di protocolli strutturata a livelli, o layers, un po' come il TCP/IP: vi sono protocolli che gestiscono l'inizio e la fine della comunicazione, la frammentazione in pacchetti e il loro riassemblaggio, il controllo delle sessioni di comunicazione, il routing, e così via. Le reti DECnet possono avvalersi di varî tipi di hardware, come Ethernet, Token Ring, FDDI su fibra ottica e X.25 . Ad ogni nodo della rete è associato un indirizzo composto da due parti, un *indirizzo di area* (*area address*), da 1 a 63, ed un *indirizzo di nodo* (*node address*), da 1 a 1023, all'interno dell'area: l'indirizzo è quindi determinato moltiplicando l'indirizzo di area per 1024, aggiungendovi l'indirizzo di nodo ed esprimendo il tutto in forma decimale (per esempio, se una stazione ha indirizzo di nodo 75 e appartiene ad un'area il cui indirizzo è 12, il suo indirizzo sarà 12363).

5.2.6 I protocolli di applicazione

Esiste un vasto repertorio di protocolli applicativi basati su TCP/IP, ognuno dettagliatamente descritto e codificato a livello internazionale in modo che tutte le macchine che lo vogliano siano in grado di utilizzarli. A solo titolo di esempio ne verranno elencati alcuni fra i più diffusi:

- *FTP* (*File Transfer Protocol*): consente di scaricare su un calcolatore un file conservato su un altro calcolatore; il protocollo fra l'altro gestisce l'autenticazione mediante la richiesta di un nome utente e di una parola chiave
- *Telnet*: consente ad un utente di collegarsi su un calcolatore remoto e di utilizzarlo come se egli fosse fisicamente presente; questo protocollo rende "invisibile" il calcolatore locale, e ogni carattere inserito viene direttamente inviato al nodo remoto (ad esclusione di alcuni caratteri di controllo interpretati invece localmente) fintantoché il programma usato per il collegamento rimane attivo; molte implementazioni del telnet includono un emulatore di terminale

– *"r"-programs*: permettono di eseguire un particolare programma su un calcolatore remoto (la "r" infatti sta per *remote*): *rexec* esegue un singolo comando sul nodo remoto, *rsh* consente di avere una shell remota, *rlogin* permette un collegamento simile a telnet; questo protocollo gestisce la richiesta di una parola chiave, ma volendo è possibile definire un insieme di nodi e utenti ritenuti sicuri, per i quali tale richiesta viene evitata (il che però può anche rappresentare un problema di sicurezza)
– *SMTP* (*Simple Mail Transfer Protocol*): questo protocollo gestisce l'invio di messaggi di posta elettronica fra due nodi; esso tra l'altro definisce la struttura che deve avere un messaggio e quali comandi devono scambiarsi i due nodi per iniziare la conversazione e permettere detto invio
– *NTP* (*Network Time Protocol*): definisce il meccanismo per distribuire una sorta di "segnale orario" fra più calcolatori che possono così sincronizzarsi fra loro
– *SNMP* (*Simple Network Management Protocol*): con questo protocollo è possibile raccogliere informazioni da svariati elementi di rete
– *HTPP* (*HyperText Transfer Protocol*): permette l'invio di un file di ipertesto (ad esempio una "pagina Web") da un nodo remoto che funge da server ad un nodo locale che si comporta da client

5.2.7 Formato dei pacchetti TCP/IP

Come si è visto, due protocolli separati sono coinvolti nella gestione dei datagrammi: il TCP è responsabile della suddivisione dei dati in pacchetti e del loro riassemblaggio, inviando di nuovo ciò che è andato perduto e rimettendo tutto nell'ordine giusto; l'IP invece è responsabile dell'instradamento (routing) dei singoli pacchetti. A prima vista può sembrare che sia il TCP a fare il grosso del lavoro, e nelle piccole reti è vero; ma su tutta Internet semplicemente far arrivare un datagramma a destinazione, tenendo traccia dei varî percorsi e gestendo le incompatibilità fra le differenti tecnologie hardware, può risultare un compito assai complesso.

L'interfaccia tra TCP e IP è abbastanza semplice: il TCP passa all'IP un datagramma con una destinazione, e l'IP lo gestisce senza sapere che relazione esso abbia col datagramma precedente o col successivo. Chiaramente però non basta far arrivare un pacchetto a destinazione: il codice di rete deve tenere traccia delle connessioni multiple da o verso uno stesso nodo, e quindi il TCP/IP deve sapere a quale specifica connessione appartenga ciascun datagramma. Questa operazione è nota come *demultiplexing*, e l'informazione necessaria a svolgerla è contenuta in una serie di *headers* (letteralmente *intestazioni*): un header è costituito semplicemente da alcuni byte aggiunti da un dato protocollo all'inizio di un datagramma al fine di poterlo identificare. Quando il pacchetto arriva a destinazione, il protocollo che lo gestisce esamina il contenuto dell'header, poi lo elimina e passa il resto del pacchetto al protocollo di livello superiore. L'esempio classico è quello della lettera e della busta: la lettera sono i dati da inviare, la busta è l'header che contiene le informazioni per farla giungere a destinazione. Sennonché questa operazione viene ripetuta più volte: è come se il mittente mettesse la lettera in una busta, l'ufficio postale mettesse quest'ultima in una busta più grande, il corriere incaricato del trasporto mettesse il tutto in una terza busta, e così via.

Si consideri un insieme di byte, ad esempio un file, da trasferire da un nodo locale ad uno remoto; per chiarezza si rappresentino questi byte come una sfilza di punti

. . . .

Il TCP divide questo flusso di byte in gruppi, i datagrammi, di grandezza opportuna (che dipende da quanti byte la rete è in grado di trasmettere con un singolo pacchetto), per esempio di 500 ottetti ciascuno

. . . .

Poi aggiunge in testa ad ognuno l'header del protocollo TCP, costituito da 20 byte e rappresentato schematicamente in Fig. 5.2 *. I *numeri di porta* (*port numbers*) vengono usati per distinguere fra loro comunicazioni differenti verso lo stesso nodo: se per esempio tre utenti diversi stanno trasferendo file verso il medesimo calcolatore, il TCP associa a ciascuno di essi un proprio numero di porta sorgente (*source port number*). Il TCP poi deve conoscere anche la porta di destinazione con la quale comunicare (*destination port number*; l'argomento delle porte sarà approfondito nel prossimo paragrafo). Ovviamente se il cal-

* Questa Figura e le due successive sono ispirate alle Figure contenute in C. H. Hedrick, *Introduction to the Internet Protocol*.

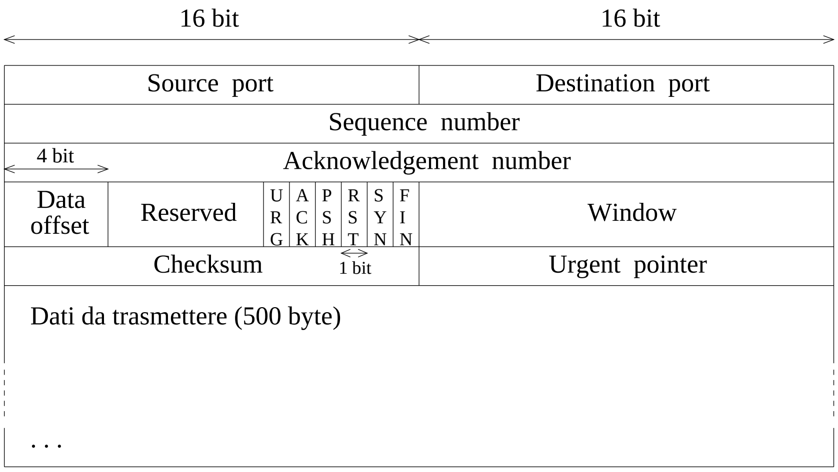

Fig. 5.2 *Header del protocollo TCP*

colatore remoto deve comunicare con uno degli utenti locali, i due numeri di porta saranno scambiati, giacché la stazione locale è ora la destinazione e quella remota la sorgente. In questo modo il TCP può tenere traccia di comunicazioni differenti. Ogni datagramma ha poi un numero sequenziale proprio (*sequence number*), in modo che a destinazione i pacchetti possano essere riassemblati nell'ordine giusto; in realtà però vengono numerati non i pacchetti ma i byte: così nell'esempio il primo datagramma sarà numerato 0, il secondo 500, il terzo 1000, eccetera.

Quando un datagramma arriva a destinazione, il nodo remoto manda indietro una sorta di "avviso di ricevimento", che non è altro che un pacchetto il cui *acknowledgment number* contiene il numero sequenziale del datagramma ricevuto: nell'esempio, un pacchetto contenente 1500 come acknowledgment number significa che il nodo remoto ha ricevuto tutti i datagrammi fino al quarto. Se il mittente non riceve risposta entro un tempo ragionevole, spedisce nuovamente lo stesso datagramma (si noti ancora una volta che è il TCP ad occuparsi di ciò: l'IP si disinteressa del pacchetto una volta partito). Vi sono poi alcuni campi, a volte singoli bit, usati specificamente per gestire la connessione e il flusso di dati. L'*urgent pointer* segnala al destinatario di saltare ad un dato byte e processarlo subito, e viene usato per trattare eventi asincroni, quali l'invio di un carattere di controllo che sospende l'output. Infine il TCP calcola un *numero di verifica*, o *checksum*, a partire da tutti i byte contenuti nel datagramma e lo aggiunge all'header; il nodo di destinazione ricalcola questo numero, e se viene diverso il pacchetto si è certamente corrotto durante il trasferimento e viene perciò scartato.

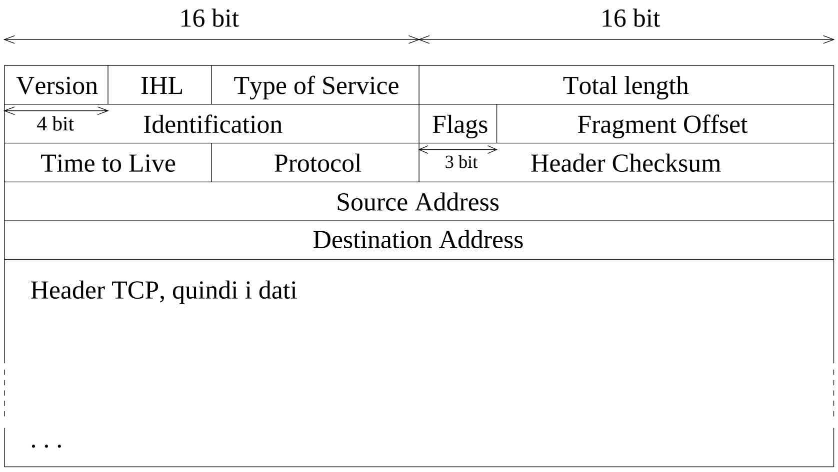

Fig. 5.3 *Header del protocollo IP*

A questo punto, se si indica con "T" l'header del TCP, la sequenza di dati che viene passata all'IP è del tipo

```
T.....   T.....   T.....   T.....   T.....   T.....
```

Insieme a ciascun datagramma il TCP passa all'IP l'indirizzo del nodo di destinazione. Questo è tutto quello che serve all'IP, il quale non si preoccupa di cosa contenga il corpo dei dati o l'header TCP (senza contare che i datagrammi potrebbero essere stati passati in consegna da altri protocolli, come UDP o ICMP, che vi avranno aggiunto in maniera analoga il loro header specifico). Compito dell'IP è trovare un percorso che colleghi il nodo mittente al destinatario e inviarvi il singolo datagramma; al fine di permettere ai gateway o ad altri intermediari di inoltrarlo, l'IP aggiunge in testa un proprio header, lungo 20 byte e schematicamente rappresentato in Fig. 5.3 . I campi più importanti sono i due indirizzi di partenza e di arrivo (*source address* e *destination address*), che sono gli indirizzi IP rispettivamente della stazione di partenza (necessario affinché l'altro nodo sappia da dove gli arrivano i datagrammi) e della stazione di arrivo. Il *numero di protocollo* (*protocol number*) specifica quale protocollo ha generato il pacchetto (TCP nell'esempio, ma potrebbe essere UDP o ICMP), cosicché l'IP del nodo destinatario sa a quale protocollo di livello superiore passare in gestione il datagramma. L'IP poi calcola un proprio checksum, diverso da quello TCP, per essere sicuro che l'header non si corrompa durante il trasferimento, altrimenti il datagramma potrebbe perdersi nella rete o finire nel posto sbagliato. Gli altri campi dell'header sono più specifici e non si entrerà nei loro

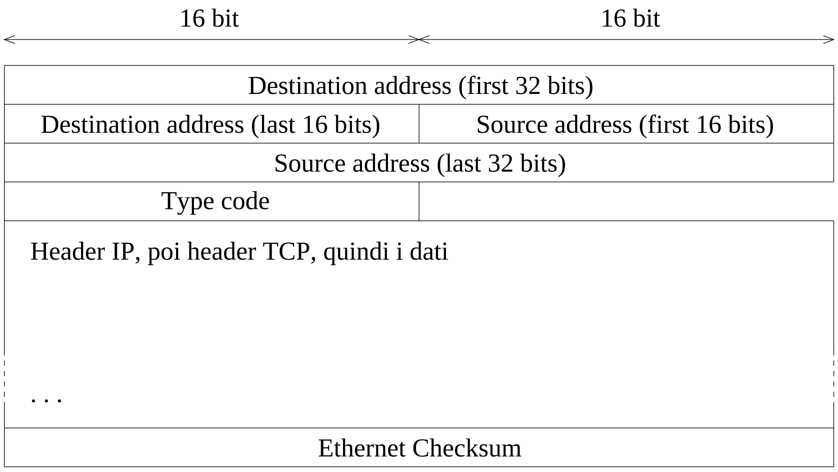

Fig. 5.4 *Header del protocollo Ethernet*

dettagli; si segnala solo il *time to live*, un numero che viene decrementato ogni volta che il datagramma passa attraverso un gateway: quando va a zero, il pacchetto viene scartato. Ciò evita che, nel caso in cui per una qualsiasi ragione si sia formato un "circuito chiuso" nel sistema, il pacchetto continui a girare in tondo all'infinito.

Se si indica con "I" l'header IP, la sequenza dei dati da trasmettere ha ora assunto la forma

```
IT.....   IT.....   IT.....   IT.....   IT.....   IT.....
```

A questo punto è possibile che non siano necessarî altri header, per esempio quando il calcolatore è collegato al destinatario o ad un gateway tramite linea telefonica. Tuttavia il protocollo di un qualsiasi altro hardware di comunicazione certamente vi aggiunge un proprio header per poter gestire i pacchetti. Essendo il più diffuso, ci si limiterà a descrivere l'header dell'Ethernet, lungo 14 byte e rappresentato in Fig. 5.4 . Si tratta di un header molto semplice, che contiene solo gli indirizzi Ethernet dei nodi mittente e destinatario e il codice del protocollo (*type code*) cui appartengono i dati trasmessi (IP, ma non solo). L'Ethernet poi calcola a sua volta un checksum sull'intero pacchetto, però lo aggiunge in coda invece che in testa: di nuovo, quando il nodo remoto riceve il pacchetto, ne ricalcola il checksum e se non corrisponde all'originale lo rigetta.

Se si rappresentano con "E" e "C" l'header e il checksum Ethernet, il flusso dei dati che viene effettivamente trasmesso nella rete assuma la forma

```
EIT.....C  EIT.....C  EIT.....C  EIT.....C  EIT.....C  EIT.....C
```

Quando i pacchetti arrivano a destinazione, tutti gli header sono rimossi in sequenza inversa. Il protocollo Ethernet rimuove header e checksum propri, controlla il type code e se corrisponde all'IP passa a quest'ultimo tutto il contenuto del pacchetto. L'IP a sua volta rimuove l'header, verifica il numero di protocollo e ne passa il contenuto al TCP se gli appartiene. Il TCP infine rimuove il proprio header, e usa il numero di sequenza per combinare insieme i datagrammi e ricostruire il flusso dei dati, per esempio il file trasmesso.

5.2.8 Le porte

Le *porte* (*ports*) possono essere considerate come i "punti di allaccio" delle connessioni di rete: se un'applicazione vuole fornire un certo servizio, si riserva una porta e aspetta i clienti (si dice che "ascolta la porta"). Una proprietà importante delle porte è che una volta stabilita una connessione fra il nodo che richiede il servizio (denominato *client*) e il nodo che lo offre (*server*), un'altra copia del programma applicativo può mettersi in ascolto sulla stessa porta: ciò consente che vi siano connessioni multiple allo stesso servizio. Il TCP è in grado di distinguere una connessione dall'altra perché esse originano o da nodi diversi o da porte diverse sullo stesso nodo.

Le porte sul server sono quindi come dei punti di incontro, che il nodo client contatta per avere la fornitura di un certo servizio; ma anche ai programmi utente viene assegnato un numero di porta. Tuttavia mentre questi ultimi possono usare numeri di porta più o meno casuali, i servizi devono avere dei numeri di porta noti a priori per poter essere contattati. Per questo una autorità centrale dirama periodicamente un documento noto come *Assigned Numbers* in cui vengono elencati i numeri di porta ufficialmente assegnati a servizi noti. Per esempio, si supponga che un utente stia lavorando su un calcolatore il cui indirizzo IP è `192.168.1.110`, e che abbia bisogno di scaricare un file conservato su un nodo remoto che ha indirizzo `192.168.2.1`. Per fare ciò egli esegue un programma interattivo chiamato `ftp`, il quale apre una connessione verso la stazione remota usando un numero di porta sorgente casuale, ad esempio 1023. Sul nodo remoto si trova un programma specializzato, l'`FTP server`, che ascolta la porta 21: quindi il programma utente `ftp` nell'aprire la connessione specifica 21 come numero di porta destinatario, perché questo è il numero ufficialmente assegnato a questo servizio e su tutte le stazioni che lo offrono esso ascolta questa specifica porta. Si noti che sono qui coinvolti due programmi, l'ftp cliente sul nodo locale, che è concepito per accettare comandi da terminale e passarli al software di rete, e l'FTP server, che è in esecuzione sul nodo remoto ed è disegnato per accettare comandi dalla connessione di rete: il programma utente può usare un numero di porta qualsiasi, ma il programma server deve usare un numero prestabilito se vuole farsi trovare.

Una qualsiasi connnessione TCP/IP è contraddistinta da quattro numeri, gli indirizzi IP dei nodi di partenza e di arrivo e i numeri di porta sorgente e destinatario (i primi due stanno nell'header IP, i secondi due nell'header TCP); non ci possono essere due connessioni con gli stessi quattro numeri. Basta comunque che uno solo dei quattro sia diverso: nell'esempio precedente, si supponga che un secondo utente stia lavorando sullo stesso calcolatore locale e abbia bisogno anch'egli di scaricare un file dallo stesso nodo remoto. In questo caso gli indirizzi IP saranno gli stessi, e anche la porta del destinatario sarà uguale, perché stanno richiedendo lo stesso servizio; tuttavia quando il programma `ftp` del secondo utente richiede una connessione, il software di rete gli assegna un numero di porta sorgente diverso da quello del primo utente, ad esempio 1022: la connessione TCP/IP di quest'ultimo sarà quindi caratterizzata dai numeri

```
192.168.1.110  192.168.2.1  1022  21
```

e quella del primo dai numeri

```
192.168.1.110  192.168.2.1  1023  21
```

Quindi il TCP/IP sia sul nodo locale che sul nodo remoto è sempre in grado di distinguere le due connessioni, esaminando il numero di porta contenuto nell'header. Ovviamente se i due utenti richiedono un servizio diverso, cambierà il numero di porta destinatario, se il secondo utente si collega ad un altro nodo cambierà l'indirizzo, e così via. Generalmente è il lato client della connessione a scegliere un numero di porta che renda unica la connessione, giacché il lato server deve avere numeri di porta predefiniti.

Si noti infine che sebbene TCP e UDP usino entrambi il meccanismo delle porte, i numeri assegnati non configgono fra loro: per esempio la porta TCP 513 è diversa dalla porta UDP 513. La distinzione in questo caso è fatta dal protocollo che è diverso.

5.2.9 Frammentazione e riassemblaggio

TCP/IP è progettato per poter funzionare con diversi tipi di rete; purtroppo tecnologie hardware diverse prevedono dimensioni dei pacchetti diverse. Apparentemente il problema si potrebbe risolvere facendo sì che IP usi sempre la dimensione di pacchetto più piccola possibile; ma ciò creerebbe dei problemi di efficienza, visto che sarebbe certo più conveniente poter usare pacchetti grandi quando la rete lo consente. Entrambi i protocolli prevedono delle soluzioni per aggirare il problema. Innanzitutto il TCP ha la possibilità di "negoziare" la dimensione dei datagrammi: quando una connessione viene aperta, entrambi i nodi si comunicano qual è la dimensione massima che ciascuno può gestire e viene poi scelta la più piccola (per convenzione ogni implementazione di protocollo di rete deve poter gestire pacchetti di almeno 576 byte). Questo però non risolve del tutto il problema, giacché i nodi estremi non possono sapere attraverso quali reti intermedie passeranno i datagrammi, e alcune di queste potrebbero essere in grado di maneggiare solo pacchetti di dimensione inferiore a quella scelta da tali nodi. Per questo l'IP è in grado di frazionare ulteriormente i datagrammi in pezzi più piccoli, ciò che viene detto *frammentazione* (*fragmentation*): ogni gateway quando passa un pacchetto verso una rete sa qual è la dimensione massima dei pacchetti che questa può sopportare e provvede a suddividerlo in frammenti se esso è troppo grande. E così l'IP dell'estremo opposto è in grado di rimettere insieme i frammenti per ricreare il pacchetto con quello che viene detto *riassemblaggio* (*reassembly*). Nell'header IP vi sono dei campi che contengono le informazioni necessarie a svolgere queste operazioni (si veda la Fig. 5.3). Si noti comunque che questa ulteriore frammentazione a livello IP è del tutto trasparente al TCP, al quale vengono sempre consegnati datagrammi interi delle dimensioni prefissate all'inizio della conversazione.

§ 5.3 Routing dei pacchetti

5.3.1 Gli indirizzi IP

Come si è accennato nel precedente paragrafo, ad ogni nodo deve essere assegnato un indirizzo IP unico su tutta la rete. Questo indirizzo è un numero a 32 bit, espresso come quattro numeri decimali, gli *ottetti* (*octets*), separati da un punto. Su una rete isolata che non comunica col mondo esterno è possibile assegnare indirizzi arbitrarî; per le reti facenti parte di Internet invece gli indirizzi vanno richiesti ad una autorità centrale, il Network Information Center (NIC).

Un indirizzo IP può essere scomposto in due parti, il *numero di rete* (*network number*), costituito dagli ottetti iniziali, e il *numero di nodo* (*host number*), costituito dagli ottetti rimanenti. Di fatto il NIC non assegna i numeri di singole macchine, ma piuttosto uno o più numeri di rete, mentre il gestore della rete può dare alle singole macchine un numero di nodo arbitrario all'interno dell'intervallo di numeri di rete assegnatogli.

Fin dall'inizio si comprese che alcune reti avrebbero potuto scegliere di organizzarsi "orizzontalmente", cioè come un'unica rete con molti nodi, mentre altre avrebbero potuto preferire una struttura "gerarchica", con molte reti interconnesse ciascuna con pochi nodi. E' chiaro che a seconda delle dimensioni della rete il numero di nodo può essere più o meno grande. Per soddisfare le diverse esigenze il cosiddetto *spazio degli indirizzi* (*address space*), cioè l'insieme di tutti i possibili indirizzi IP, è stato diviso in classi di reti, a seconda del numero di bit assegnati al numero di nodo. Negli indirizzi di *Classe A* solo il primo ottetto è riservato al numero di rete e gli ultimi tre al numero di nodo: tale classe comprende gli indirizzi di rete da `1.0.0.0` a `126.0.0.0` . Gli indirizzi di *Classe B* invece sono divisi in parti uguali, i primi due ottetti al numero di rete e i secondi due al numero di nodo; a questa classe appartengono tutti gli indirizzi di rete da `128.0.0.0` a `191.255.0.0` . Gli indirizzi di *Classe C* assegnano i primi tre ottetti al numero di rete e solo l'ultimo ottetto al numero di nodo, e comprendono tutti gli indirizzi di rete da `192.0.0.0` a `223.255.255.0` . Infine vi sono le classi *D*, *E* e *F* che comprendono gli indirizzi di rete da `224.0.0.0` a `254.0.0.0` , i quali sono o indirizzi sperimentali o riservati per usi futuri e non sono specifici di una rete. In questo modo sono possibili 126 grandi reti, ognuna delle quali può indirizzare oltre 16.6 milioni di nodi, 16320 reti di media grandezza, ognuna delle quali può ospitare 65024 nodi, e oltre 2 milioni di reti piccole con 254 nodi ciascuna. Per esempio, l'indirizzo IP `192.168.1.15` è un indirizzo di Classe C e identifica una stazione con numero di nodo `15` all'interno della rete il cui indirizzo è `192.168.1.0` .

Si noti che non tutti i valori sono ammessi come numero di nodo: infatti i numeri di nodo `0` e `255` sono riservati. Per convenzione un indirizzo in cui tutti i bit del numero di nodo sono zero indica la rete stessa, mentre un indirizzo in cui tutti i bit del numero di nodo sono uno si riferisce a tutti i nodi della rete (e per questo viene detto *broadcast address*). Così `192.168.1.255` non è un numero di nodo valido, ma specifica contemporaneamente tutti i nodi della rete `192.168.1.0` .

Ci sono pure due numeri di rete riservati, `0.0.0.0` e `127.0.0.0` : il primo è detto *default route* ed è connesso al modo in cui IP instrada i datagrammi, il secondo è detto *loopback address* ed è riservato al traffico locale di una stazione. Un solo numero di nodo è ammesso, `127.0.0.1` , e viene assegnato ad una interfaccia fittizia, detta *loopback interface*, che si comporta come un circuito chiuso: ogni pacchetto IP inviato a questo indirizzo viene restituito indietro come se provenisse da una rete esterna. Questo permette di sviluppare, testare e utilizzare il software di rete anche su nodi non fisicamente connessi ad una rete.

5.3.2 Il routing dei pacchetti

Il compito del protocollo IP è di trovare un percorso per inviare un pacchetto a destinazione. Per decidere quale strategia usare il software confronta l'indirizzo IP del nodo di arrivo con il proprio indirizzo, determinando se entrambi i nodi mittente e destinatario appartengano alla stessa rete. Se è questo il caso, il datagramma può essere recapitato dalla rete locale, altrimenti occorre affidarsi ad un gateway. Da qui la necessità che reti fisicamente separate abbiano indirizzi di rete diversi.

Si supponga che il nodo `alpha` avente indirizzo IP `192.168.1.10` voglia inviare un datagramma al nodo `beta` di indirizzo `192.168.1.20` (Fig. 5.5). Il software IP si accorge che il nodo destinatario appartiene alla stessa rete `192.168.1.0` e decide che può spedirgli direttamente il pacchetto. Tuttavia per fare ciò gli occorre il suo indirizzo Ethernet, mentre ora il nodo mittente ne conosce solo l'indirizzo IP*. Interviene allora un altro protocollo separato, non facente parte della famiglia TCP/IP, detto *Address Resolution Protocol* (*ARP*), il quale è in grado di determinare l'indirizzo Ethernet associato ad un certo indirizzo IP. Per fare ciò esso invia sulla rete un pacchetto con un indirizzo particolare, detto *broadcast address*, con tutti i bit a 1, e contenente l'indirizzo IP da ricercare.

---

* In questa discussione ci si limita al caso in cui le reti locali siano basate sulla tecnologia Ethernet, essendo questa la più diffusa; meccanismi analoghi esistono per gli altri tipi di hardware.

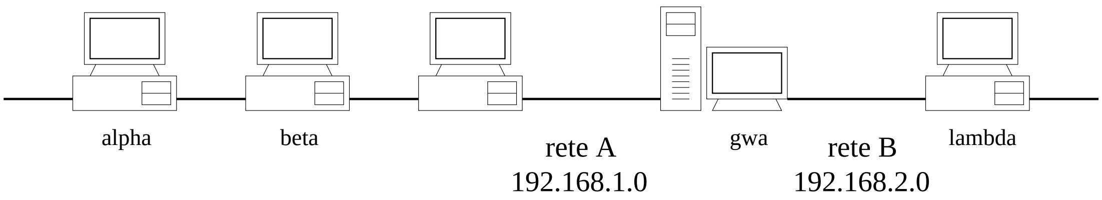

Fig. 5.5 *Esempio di routing fra due reti*

Per convenzione tutti i nodi devono prestare attenzione ai pacchetti di broadcast: ognuno quindi legge l'indirizzo IP e lo confronta col proprio. Il nodo cui tale indirizzo appartiene, `beta` nell'esempio, risponde al mittente inviandogli il proprio indirizzo Ethernet: in questo modo `alpha` viene a conoscenza dell'indirizzo che gli serve, e può formare l'header del pacchetto, il quale viene poi immesso nella rete e ricevuto da `beta`. Per evitare di dover continuamente richiedere questa informazione, il nodo mittente conserva l'indirizzo Ethernet corrispondente ad un certo indirizzo IP in una tabella temporanea detta *ARP cache*, cosicché quando deve inviare un nuovo pacchetto conosce già l'indirizzo. Tuttavia questo dato non viene mantenuto in modo permanente, visto che l'indirizzo Ethernet potrebbe cambiare, per esempio se la scheda di rete si guasta e viene sostituita: dopo un certo tempo quindi l'informazione "scade" e quando serve inviare un altro pacchetto viene generata una nuova richiesta ARP.

Esiste anche il protocollo inverso, detto *Reverse Address Resolution Protocol* (*RARP*), in grado di determinare l'indirizzo IP associato ad un indirizzo Ethernet. Esso trova applicazione in casi particolari, per esempio quando una macchina priva di disco deve avviarsi da rete scaricando il sistema operativo da un server apposito.

Se invece il nodo destinatario appartiene ad una rete diversa, il problema si complica. Si supponga ad esempio che il nodo `alpha` voglia inviare un datagramma al nodo `lambda` di indirizzo IP `192.168.2.100` : ora il software si rende conto che il destinatario non sta sulla stessa sua rete e che quindi deve fare ricorso ad un gateway. Quest'ultimo non è nient'altro che una stazione collegata contemporaneamente a due o più reti e in grado di scambiare pacchetti fra due qualsiasi di esse. Di conseguenza un gateway possiede più indirizzi IP, uno per ogni rete cui è connesso (a rigore un indirizzo IP non è assegnato ad un nodo bensì alla sua interfaccia di rete: pertanto un gateway avendo più interfacce ha anche più indirizzi, uno per ciascuna di esse). Nel caso più semplice e più comune ogni rete locale ha un solo gateway che la collega ad una rete più grande; tuttavia è possibile per una rete avere diversi gateway.

Per sapere come comportarsi ogni nodo mantiene una tabella, detta *routing table*, in base alla quale il protocollo IP decide come inoltrare un pacchetto. Per un nodo della rete A di Fig. 5.5 come `alpha` questa tabella è molto semplice, del tipo

Rete	**Gateway**	**Interfaccia**
127.0.0.0	-	*lo*
192.168.1.0	-	*eth0*
0.0.0.0	192.168.1.1	*eth0*

dove `192.168.1.1` è l'indirizzo del gateway `gwa` nella rete A. La prima riga indica al software come comportarsi di fronte ai pacchetti indirizzati al loopback address: essi devono passare attraverso la fittizzia interfaccia di loopback *lo*, e per fare ciò non c'è bisogno di gateway. La seconda riga invece serve a inoltrare i pacchetti indirizzati ai nodi della rete locale cui appartiene `alpha`: anche per essi non vi è bisogno di gateway, e i pacchetti passano per l'interfaccia Ethernet *eth0*. La terza riga infine indica il destino di tutti gli altri pacchetti (è la cosiddetta *default route*): essi vanno affidati al gateway di indirizzo `192.168.1.1` e anch'essi devono passare attraverso l'interfaccia *eth0*.

Nel momento in cui vuole inviare un pacchetto verso `lambda`, il nodo `alpha` consulta la propria routing table e scopre che per fare questo deve affidare tale pacchetto a `gwa`: ne determina quindi l'indirizzo Ethernet con una richiesta ARP, costruisce l'header relativo e lo invia come un qualunque altro pacchetto Ethernet (d'altra parte, il gateway da questo punto di vista è come un qualsiasi altro nodo della rete A). Fatto ciò, `alpha` si disinteressa del pacchetto: per lui è stato inviato, sarà poi compito del TCP verificare che sia arrivato a destinazione o vada nuovamente spedito.

Il gateway `gwa` ha due indirizzi IP, uno per l'interfaccia verso la rete A, `192.168.1.1` nell'esempio, ed uno per l'interfaccia verso la rete B, ad esempio `192.168.2.1` . Quando `gwa` riceve il pacchetto di `alpha`, ne esamina l'indirizzo di destinazione e consulta la propria routing table, che però è un po' più complessa, del tipo

Rete	**Gateway**	**Interfaccia**
127.0.0.0	-	*lo*
192.168.1.0	-	*eth0*
192.168.2.0	-	*eth1*
⋮	⋮	⋮

Poiché il nodo `lambda` appartiene ad una rete cui il gateway è direttamente collegato attraverso l'intefaccia *eth1*, richiede l'indirizzo Ethernet del nodo destinatario e poi gli invia il pacchetto attraverso la rete locale B.

I gateway però mantengono informazioni anche su come raggiungere reti cui non sono fisicamente connessi. Se per esempio il nodo `alpha` vuole inviare un datagramma al nodo `sigma` di indirizzo `192.168.10.55` (Fig. 5.6), lo passa sempre al suo gateway `gwa`. Quest'ultimo consulta ancora la propria tabella di routing, dove si trovano altre informazioni, del tipo

Rete	**Gateway**	**Interfaccia**
127.0.0.0	-	*lo*
192.168.1.0	-	*eth0*
192.168.2.0	-	*eth1*
192.168.10.0	192.168.2.10	*eth1*
⋮	⋮	⋮
0.0.0.0	192.168.2.254	*eth1*

dove `192.168.2.10` è l'indirizzo IP del gateway `gwc` nella rete B. In base a questa tabella i pacchetti indirizzati alla rete `192.168.10.0` vanno inoltrati al gateway `192.168.2.10`: pertanto `gwa` ne richiede l'indirizzo Ethernet e poi gli spedisce il pacchetto di `alpha`.

A sua volta `gwc` ricevuto il pacchetto ne legge l'indirizzo di destinazione e poi consulta la propria tabella, che sarà analoga a quella di `gwa`: dato che il nodo di arrivo `sigma` appartiene ad una rete cui esso è fisicamente connesso, ne determina l'indirizzo Ethernet e quindi gli invia il pacchetto attraverso la rete C.

Anche i gateway possono avere una default route, da usare quando non sanno come recapitare un pacchetto a loro affidato. Se per esempio `alpha` volesse inviare un datagramma al nodo di indirizzo `172.16.5.104`, poiché esso non appartiene alla sua rete lo passerebbe al proprio gateway `gwa`. Questi a sua volta potrebbe non avere nella propria

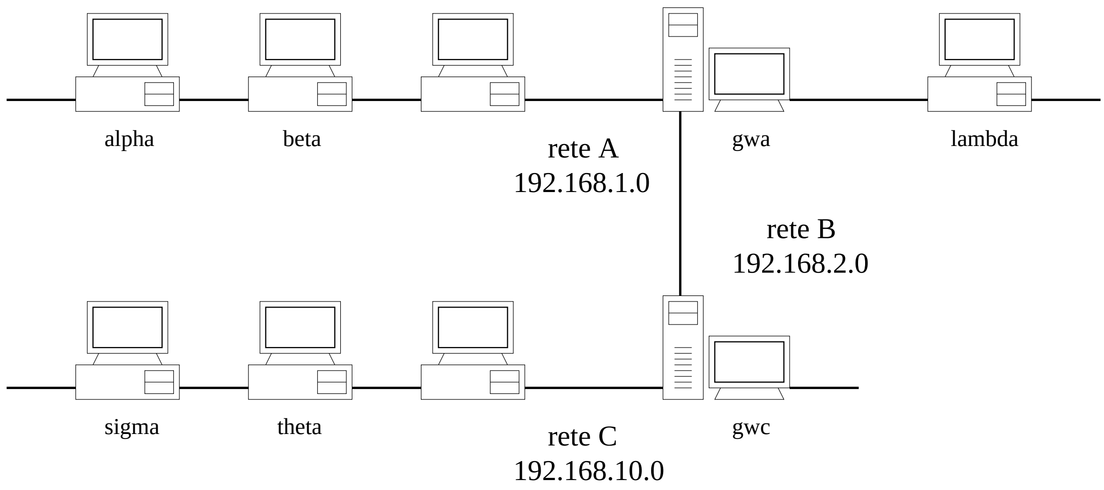

Fig. 5.6 *Esempio di routing fra tre reti*

tabella di routing alcuna informazione su questa rete: in tal caso invierebbe il pacchetto al gateway di indirizzo `192.168.2.254`, che o conosce come farlo arrivare a destinazione oppure lo inoltra ad un altro gateway ancora.

In sostanza quindi il compito principale di un gateway è di trovare una strada per un pacchetto. Non è necessario che un gateway sappia esattamente dove inviarlo: al contrario, spesso esso sa solo dove vanno girati i pacchetti il cui indirizzo di destinazione appartiene ad un certo gruppo più o meno ampio di indirizzi. Per fare ciò esso consulta la propria tabella di routing, che è in genere più complessa di quanto mostrato nell'esempio.

Nelle reti più piccole le tabelle di routing possono essere costruite "a mano": in questo caso le informazioni in esse contenute sono di tipo statico e non soggette a cambiamento (i nodi che non sono gateway possiedono generalmente delle tabelle statiche). Nelle reti più grandi invece queste tabelle vengono modificate dinamicamente e mantenute aggiornate da processi presenti sui gateway: questi si scambiano informazioni di routing in modo da conoscere sempre il percorso migliore fra le reti di cui essi sono membri. A tal fine esistono diversi protocolli per lo scambio di informazioni di routing. Nel caso di tabelle dinamiche, ad ogni percorso verso una rete di destinazione viene associato un valore basato sul numero di gateway che un datagramma deve attraversare per raggiungerla: per inviare un pacchetto verso una data rete viene quindi scelto il percorso che richiede il minor numero di gateway intermedi.

§ 5.4 Il sistema dei nomi di dominio

Per poter inviare un datagramma ad un nodo remoto è necessario conoscerne l'indirizzo IP. Tuttavia sarebbe certo molto scomodo dover ricordare a memoria gli indirizzi numerici di tutti i nodi che interessano. Ciò non è necessario in quanto è possibile assegnare dei "nomi" ai nodi in rete: è poi compito del protocollo TCP/IP trovare l'indirizzo IP corrispondente ad un certo nome (operazione nota come *host name resolution*). A questo scopo esistono delle funzioni di libreria che consentono ad una applicazione di tradurre un nome di nodo in un indirizzo e viceversa.

Inizialmente ogni nodo di una rete conservava su un file locale una tabella che faceva corrispondere il nome dei nodi con i loro indirizzi. Questo metodo poteva funzionare su piccole reti locali, ma con il crescere del numero di nodi in rete la sua gestione divenne insostenibile, giacché l'amministratore di ciascuna macchina doveva continuamente tenere aggiornato il file. Una parziale soluzione al problema fu il *Network Information System* (*NIS*) della Sun Microsystems, nel quale le tabelle con i nomi e gli indirizzi dei nodi sono conservate in una macchina server centrale che è in grado di distribuirle alle macchine suoi satelliti. Tuttavia pure con questo sistema molti dei problemi rimanevano nella gestione di grandi insiemi di nodi. Anche a livello di Internet queste informazioni erano inizialmente conservate in un unico file mantenuto presso il Network Information Center, dal quale tutti i nodi partecipanti alla rete lo scaricavano periodicamente. Questo metodo presentava due problemi fondamentali, che si fecero insostenibili con la rapida crescita del numero di nodi in rete: da un lato chiunque voleva dare un nome alla sua stazione doveva registrarsi presso questa autorità centrale, la quale fra l'altro controllava che non ci fossero due macchine con lo stesso nome; in secondo luogo le dimensioni di questo file crebbero molto rapidamente e scaricarlo dal server centrale del NIC rappresentava un carico molto pesante per la rete in generale e per il NIC in particolare. Entrambi questi problemi vennero risolti nel 1984 con l'introduzione del sistema dei nomi di dominio.

Il *sistema dei nomi di dominio* (*Domain Name System* o *DNS*) organizza i nomi dei nodi in modo gerarchico. Lo "spazio dei nomi", cioè l'insieme di tutti i nomi dei nodi, viene diviso in *dominî* (*domains*): un dominio è un insieme di nodi o di siti aventi una qualche attinenza fra loro, per esempio perché formano una rete propria o perché appartengono

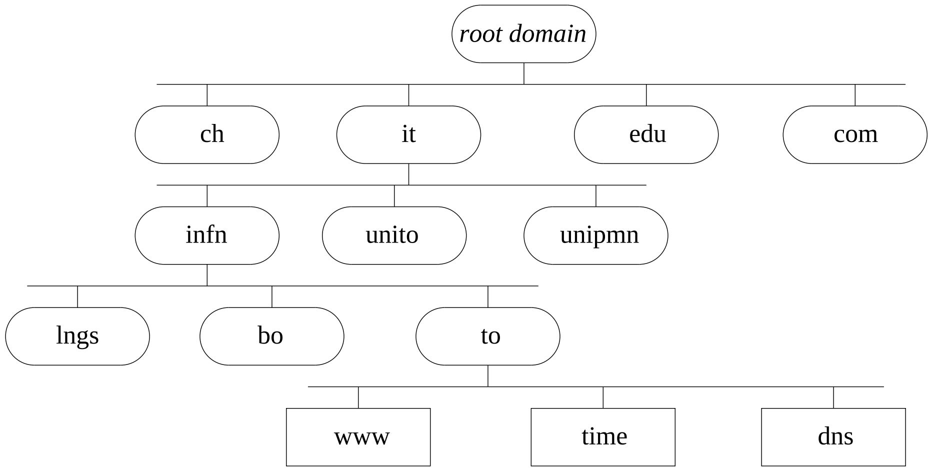

Fig. 5.7 *La gerarchia dei nomi di dominio*

ad una certa organizzazione o ancora semplicemente perché sono geograficamente vicini. L'intero spazio dei nomi forma un unico dominio, detto *dominio radice* (*root domain*); esso viene suddiviso in dominî di primo livello (*top-level domains*), i quali a loro volta possono essere suddivisi in sottodominî di secondo livello, terzo livello, e così via in una struttura ad albero (Fig. 5.7).

In questo sistema il nome di una stazione è costituito da due parti, il nome vero e proprio del nodo (*host name*) e il nome del dominio cui esso appartiene (*domain name*); insieme essi formano il cosiddetto *fully qualified domain name*, in cui i nomi del nodo e del dominio (e degli eventuali sottodominî) sono separati da punti. Per esempio, il nome `cicladi.unipmn.it` appartiene al nodo `cicladi` che fa parte del dominio `unipmn.it`.

Il sistema dei nomi di dominio risolve così un primo problema, quello dell'unicità dei nomi dei nodi: è sufficiente infatti che il nome di un nodo sia unico all'interno del sottodominio cui appartiene perché diventi automaticamente unico su tutta la rete.

Il dominio radice è stato suddiviso in dominî nazionali, cosicché in genere ogni Nazione usa un proprio dominio di primo livello; il nome di ciascuno di essi è costituito da due lettere che indicano lo Stato secondo uno standard internazionale: ad esempio, `it` indica il dominio assegnato all'Italia, `fr` quello assegnato alla Francia, `ch` quello della Svizzera, e poi `de` per la Germania, `uk` per il Regno Unito, `es` per la Spagna, `pt` per il Portogallo, `jp` per il Giappone, `mx` per il Messico, `au` per l'Australia, e molti altri ancora. Gli Stati Uniti fanno eccezione: sebbene esista anche il nome di dominio `us`, la stragrande maggioranza dei nodi è stata suddivisa in dominî di primo livello indicati da tre lettere, che sono `edu`, comprendente in genere reti di università ed enti di studio, `com`, per siti commerciali, `org`, per altre organizzazioni non commerciali, `net`, che comprende gateway e nodi amministrativi della rete, `mil`, per le istituzioni militari, e `gov`, per le istituzioni governative (questa suddivisione riflette il fatto che storicamente il sistema dei nomi di dominio fu introdotto per primo nel Nord America).

All'interno di ciascun dominio di primo livello l'autorità che lo gestisce è libera di organizzarlo come preferisce, assegnando nomi e creando sottodominî come ritiene più opportuno; e a sua volta i responsabili dei sottodominî hanno la stessa libertà all'interno della propria area di amministrazione. Infatti il secondo grande vantaggio del sistema dei nomi di dominio è di delegare l'autorità su un sottodominio ai suoi responsabili, i quali sono liberi di creare sottoaree e di assegnare nomi e indirizzi IP all'interno delle proprie reti senza alcuna interferenza esterna. In tal modo viene risolto il primo grande problema del NIC: non è più necessaria una autorità centrale cui comunicare ogni variazione a nomi e indirizzi IP delle proprie reti. Questa autorità si limita ad assegnare un gruppo di indirizzi alle istituzioni di ciascun dominio, e queste a loro volta le suddividono nelle proprie sottoaree. Ad esempio, i gestori del dominio `it` hanno creato un sottodominio riservato ai calcolatori dell'Istituto Nazionale di Fisica Nucleare, denominato `infn.it`, e ne hanno delegato ai suoi responsabili la gestione; poiché questo Istituto ha molte sedi in Italia, questi gestori hanno creato un sottodominio per ciascuna sede, ad esempio `to.infn.it` per la sede di Torino, `mi.infn.it` per quella di Milano, `lngs.infn.it` per i Laboratori Nazionali del Gran Sasso, eccetera. Ogni sede ha dei responsabili di rete che la gestiscono, assegnando in piena libertà nomi e indirizzi IP ai propri calcolatori.

Se un nodo appartiene ad un certo dominio nazionale, ciò tuttavia non significa necessariamente che tale macchina sia fisicamente localizzata in quello Stato: significa solo che il suo nome è stato registrato presso l'autorità di quel dominio. Per esempio, una ditta italiana potrebbe avere delle succursali in altri Stati esteri, e avere comunque tutti i nodi registrati nel dominio `it` per facilità di identificazione. Questo è particolarmente vero per i dominî `com` e `org` (e in parte anche per `edu`): molto spesso i siti il cui fully qualified domain name finisce per `.com` o `.org` appartengono ad organizzazioni e società non localizzate negli Stati Uniti, ma che hanno chiesto alle autorità di rete statunitensi di essere registrati nel loro dominio per comodità (per esempio perché è oramai entrato nell'uso comune pensare che una compagnia commerciale debba avere un sito il cui nome finisce per `.com`).

Si immagini che una applicazione sul nodo `alpha` debba inviare un pacchetto ad un nodo remoto di cui conosce il nome (cioè il fully qualified domain name) ma non l'indirizzo IP. Ciascuna macchina possiede un elenco di nodi che sono in grado di fornire l'indirizzo associato ad un nome, i cosiddetti *name servers* o *DNS servers*. Allora il software di rete invia una richiesta ad uno di essi: se il server ha questa informazione la passa al nodo che l'ha interpellato. Se non possiede tale informazione, il server DNS inizia una *ricerca iterativa* (*iterative query*). Si supponga ad esempio che al server `dnssrv` sia stato chiesto di trovare l'indirizzo IP del nodo `www.to.infn.it`: esso allora interroga uno dei name server per il dominio radice con una richiesta di indirizzo per quel nodo. Il server del dominio radice riconosce che il nodo appartiene ad un dominio che non è sotto la sua autorità ma sotto quella del dominio `it`, e quindi invia a `dnssrv` una lista di nomi e indirizzi di DNS server per il dominio `it`. Il server `dnssrv` allora si rivolge ad uno di essi chiedendo di nuovo l'indirizzo IP desiderato; ma a somiglianza del primo anche questo secondo server si accorge che il nodo appartiene ad un sottodominio che non è di sua competenza, e restituisce a `dnssrv` una lista di name server per il dominio `infn.it`. Di nuovo il server invia una richiesta ad uno di essi, e riceve come risposta un elenco di server per il dominio `to.infn.it`. Quando poi `dnssrv` interroga uno di questi ultimi, esso si accorge che il nodo `www.to.infn.it` appartiene in effetti al proprio dominio, ne cerca l'indirizzo IP nei proprî elenchi e lo restituisce a `dnssrv`, il quale finalmente lo passa a `alpha`.

Potrebbe sembrare che questo meccanismo generi un grande traffico sulla rete, ma in realtà pensando ai milioni e milioni di nodi esistenti il traffico è molto meno di quanto sarebbe necessario per scaricare un unico enorme file da un unico server centrale: infatti in questo modo le richieste vengono distribuite su più server, e ognuno di essi restituisce solo una piccola quantità di informazione. Inoltre per abbreviare i tempi di ricerca i name server come `dnssrv` conservano in una tabella temporanea gli indirizzi dei DNS server di dominio già consultati, in modo da poter saltare alcuni passaggi intermedi: se per esempio in seguito riceve una richiesta di indirizzo IP per il nodo `www.mi.infn.it`, `dnssrv` si rivolge direttamente al DNS server del dominio `infn.it`. Naturalmente queste informazioni vengono mantenute solo temporaneamente, dopo un certo periodo diventano obsolete e alla prossima occasione il server le chiede di nuovo al dominio di livello superiore.

Il sistema dei nomi di dominio costituisce una sorta di "database distribuito", che risolve il secondo grande problema del NIC, cioè la concentrazione delle informazioni in un unico luogo. Esso inoltre, come si è già detto, lascia una grande autonomia ai gestori locali, che possono modificare nomi e numeri IP in piena libertà. Ad ogni livello ci sono più di un DNS server, sia per suddividere il carico di lavoro, sia per supplire alle eventuali indisponibilità di uno di essi.

Bibliografia

O. Kirch, *The Linux Network Administrator's Guide*, The Linux Documentation Project, 1994

C. H. Hedrick, *Introduction to the Internet Protocol*, Center for Computer and Information Services, The Rutgers University, 1988

The nice thing about standards
is that there are so many of them to choose from.

Andrew S. Tanenbaum

Appendice A

Il codice ASCII

Appendix:
A portion of a book, for which nobody yet has discovered any use.
Linux "fortune"

Nella stragrande maggioranza dei calcolatori ogni carattere occupa un byte, il che rende possibile rappresentare fino a 256 caratteri distinti. Esistono diversi standard di conversione che associano ad ogni byte un carattere, ma quello di gran lunga più diffuso è lo standard *ASCII* (per *American Standard Code for Information Interchange*). Occorre però eliminare una ambiguità: il vero standard ASCII consiste di soli 128 caratteri, associati ai codici da 0 a 127; i restanti 128 caratteri compresi fra 128 e 255 costituiscono il cosiddetto *ASCII esteso*. Ora mentre la prima metà è universalmente accettata non solo dai calcolatori ma da molte altre apparecchiature elettroniche, la seconda può variare in ambiti diversi: spesso quindi allo stesso codice macchine diverse associano caratteri diversi, legati ad esempio all'alfabeto locale o a necessità specifiche. Tuttavia non sempre si fa distinzione fra ASCII vero e proprio e ASCII esteso, ma l'intero set viene chiamato semplicemento "codice ASCII", il che può ingenerare confusione. In Tab. A.1 è riportata la conversione fra codice e carattere per la prima parte del set.

I codici da 32 a 126 rappresentano caratteri ordinari, cioè lettere, cifre, segni di punteggiatura e segni diacritici (si noti che la rappresentazione delle cifre da 0 a 9 non corrisponde al loro valore numerico); il codice 32 corrisponde ad uno spazio bianco. Le lettere maiuscole e minuscole hanno codici diversi e ciò facilita il compito ai programmi di manipolazione dei testi (per esempio quando devono ordinare delle parole in ordine alfabetico o devono cercare una parola in un testo). I codici 0 e 127 sono particolari: corrispondono

Tabella A.1 Codice ASCII standard.

Carattere	Codice	Carattere	Codice	Carattere	Codice	Carattere	Codice
NUL	0		32	@	64	`	96
SOH	1	!	33	A	65	a	97
STX	2	“	34	B	66	b	98
ETX	3	#	35	C	67	c	99
EOT	4	$	36	D	68	d	100
ENQ	5	%	37	E	69	e	101
ACK	6	&	38	F	70	f	102
BEL	7	´	39	G	71	g	103
BS	8	(	40	H	72	h	104
HT	9	)	41	I	73	i	105
JLF	10	$\ast$	42	J	74	j	106
VT	11	+	43	K	75	k	107
FF	12	,	44	L	76	l	108
CR	13	−	45	M	77	m	109
SO	14	.	46	N	78	n	110
SI	15	/	47	O	79	o	111
DEL	16	0	48	P	80	p	112
DC1	17	1	49	Q	81	q	113
DC2	18	2	50	R	82	r	114
DC3	19	3	51	S	83	s	115
DC4	20	4	52	T	84	t	116
NAK	21	5	53	U	85	u	117
SYN	22	6	54	V	86	v	118
ETB	23	7	55	W	87	w	119
CAN	24	8	56	X	88	x	120
EM	25	9	57	Y	89	y	121
SUB	26	:	58	Z	90	z	122
ESC	27	;	59	[	91	{	123
FS	28	<	60	\	92	$\vert$	124
GS	29	=	61	]	93	}	125
RS	30	>	62	ˆ	94	˜	126
US	31	?	63	_	95	NUL	127

a caratteri nulli, ovvero caratteri che vengono semplicemente ignorati da chi li riceve. Essi vengono comunemente impiegati per mantenere attiva una linea mediante la loro trasmissione quando non viene inviato alcun dato.

I primi 32 caratteri ASCII hanno invece funzioni speciali: sono dei *caratteri di controllo*. Questi codici vengono usati in tutte le operazioni di trasmissione di dati (per esempio verso una stampante o un modem) per controllarne il flusso: essi stabiliscono "come" i dati debbano essere trasmessi. La loro funzione esplicita cambia in realtà a seconda del sistema che li riceve, poiché ognuno li interpreta secondo il proprio hardware. Data questa variabilità ci si limiterà a segnalare l'uso dei codici più importanti senza scendere nei dettagli. I caratteri *Backspace* (codice 8), *Tab* (9), *Line Feed* (10) e *Form Feed* (12) vengono comunemente usati per istruire le stampanti sul formato dei testi; il codice 7 (BEL) produce un segnale acustico, mentre il codice 3 può essere utilizzato per interrompere il programma in corso. Altri caratteri come l'*Acknowledge* (codice 6), il *Negative Acknowledge* (21) e il *Synchronize* (22) trovano largo impiego nelle comunicazioni fra calcolatori. Tutti questi caratteri possono essere ottenuti anche premendo il tasto *Control* insieme ad un altro tasto: così ad esempio BEL si produce anche con *Control–G*, l'interruzione con *Control–C* e il Backspace con *Control–H*. Spesso però questi codici non sono sufficienti e allora si usa il carattere *Escape* (27) come prefisso per altri codici: quando un componente (una stampante, un modem, ma anche un programma) riceve un carattere di Escape interpreta il carattere successivo come un particolare comando.

Come si vede anche dalla Tabella A.1 i primi 128 caratteri del codice ASCII contengono tutti i caratteri necessari per scrivere testi in inglese (americano). I successivi 128 sono utilizzati per rappresentare caratteri specifici di altri alfabeti (prevalentemente europei), anche se la conversione da codice a carattere non è altrettanto rigidamente definita e talora sistemi diversi associano una stessa lettera a codici differenti o viceversa lettere differenti allo stesso codice. Fra i caratteri generalmente presenti nei varî codici estesi vi sono tutte le vocali accentate (come è, é, ò, ì, e anche ê e î) e con dieresi (come ä e ë), la cedilla (ç) e la doppia esse tedesca (ß), i dittonghi æ e œ e i punti interrogativo ed esclamativo capovolti (¿ e ¡) usati in spagnolo. Vi sono anche alcuni semplici simboli matematici come $\pm$, ½ , ¼ , ², ³ e i simboli valutari della sterlina £, dello yen ¥ e del cent ¢ (il simbolo del dollaro $ fa invece parte dell'ASCII standard).

Appendice B

Introduzione allo Unix

Computers are good at following instructions, but not at reading your mind.

Donald E. Knuth, "The TEXBook"

In questa appendice vengono riassunti i principali comandi Unix; per tutti i concetti di carattere generale si rimanda il lettore al Capitolo 3, che si suppone già noto. Data la mancanza di uniformità fra le interfacce grafiche ci si limita ad esporre i comandi usati in una sessione da terminale alfanumerico (di cui per altro ogni interfaccia grafica ha uno o più emulatori). Naturalmente non si ha la benché minima pretesa di essere esaustivi: sono qui forniti quei primi elementi che consentono ad un utente inesperto di prendere confidenza con questo sistema operativo.

Molti comandi Unix hanno un significato mnemonico, anche se talora un po' criptico, e che tuttavia può aiutare a ricordarli. Spesso tale significato verrà riportato.

Negli esempi che seguono verrà indicato con carattere `tipografico` i messaggi e le risposte del calcolatore e con carattere *inclinato* i comandi inseriti dall'utente.

Apertura di una sessione

Come avviene in tutti i sistemi multiutente, per poter accedere ad un sistema Unix occore identificarsi. Per questo viene richiesto un nome utente (*username*) ed una parola chiave (*password*). Se il collegamento avviene da terminale alfanumerico (direttamente collegato al calcolatore oppure via `telnet`) si avrà qualcosa di simile

```
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.14-5.0 on an i686
login:   auser
Password: ThEpass
```

B.1 *Esempio di finestra di identificazione dell'utente*

(in realtà la parola chiave non appare sullo schermo quando viene inserita). Analogamente avviene se ci si collega in locale usando una interfaccia grafica, la quale, come esemplificato in Fig. B.1 , presenterà una finestra di identificazione.

Il sistema a questo punto effettua alcune operazioni di preparazione (come definizione di variabili globali e di variabili di utente), quindi avvia la shell, che si presenta con una stringa, detta *prompt*: è ora possibile inviare comandi.

Il comando *exit* è usato per terminare la sessione

```
$  exit
Connection closed by foreign host.
```

Il comando *who* può essere usato per vedere quali utenti sono collegati sul calcolatore

```
$  who
auser    tty1     Mar  4 16:20
auser    pts/0    Mar  4 16:20
```

esso mostra il nome degli utenti, il terminale da cui sono collegati e la data e l'ora in cui il collegamento è avvenuto. Il comando *w* è analogo, ma mostra anche l'eventuale macchina remota da cui l'utente è collegato, quale comando sta eseguendo in quel momento e quanto tempo ha passato senza far nulla (*idle*) oppure occupando la macchina

```
$  w
6:38pm  up  9:50,  2 users,  load average:  0.01, 0.02, 0.00
USER     TTY      FROM              LOGIN   IDLE   JCPU   PCPU  WHAT
auser    tty1     -                 4:20pm  2:17m 14.26s  0.02s  emacs
auser    pts/0    -                 4:20pm  0.00s 27.04s  0.02s  w
```

La prima riga fornisce delle informazioni sulla macchina che possono essere ottenuto pure col comando *uptime*, e precisamente l'ora, da quanto tempo il calcolatore è in funzione (*up*), quanti utenti sono connessi, e il carico medio della macchina negli ultimi 1, 5 e 15 minuti

```
$  uptime
6:39pm  up  9:51,  2 users,  load average:  0.04, 0.02, 0.00
```

E' buona norma cambiare di frequente la propria parola chiave. Nei sistemi in cui le password sono conservate localmente si usa il comando *passwd* per effettuare il cambiamento; invece nei sistemi in cui esse sono conservate in un server centrale e distribuite tramite NIS, come avviene nel Laboratorio di Informatica per Fisici, occorre usare il comando *yppasswd* che comunica la nuova parola chiave al server centrale. Entrambi i comandi richiedono la password attuale e poi la nuova; quest'ultima va inserita due volte onde evitare errori di battitura: ad esempio

```
$  yppasswd
Changing NIS password for auser
Old NIS password:
New password:
Retype new password:
Password successfully changed on server
```

(al solito le parole chiave non appaiono sullo schermo quando vengono scritte).

Directory e files

Quando l'utente si collega si trova nella sua directory di riferimento (*home directory* o *login directory*). Per cambiare directory si usa il comando *cd* (per *change directory*)

```
$  cd  nuova-directory
```

dove *nuova-directory* può essere sia una directory assoluta, cioè riferita alla directory radice

```
$  cd  /usr/src/linux
```

sia una directory relativa, cioè riferita alla directory corrente: se `dati` è una sottodirectory della directory corrente, col comando

```
$  cd dati
```

si scende di un livello. Il comando *cd* da solo riporta nella propria directory di login (alla quale ci si può riferire anche con simbolo “tilde” ˜).

Per conoscere la directory corrente in cui ci si trova c'è il comando *pwd* (per *print working directory*)

```
$  pwd
/home/users/auser
$
```

Per vedere i files contenuti in una directory si usa il comando *ls* (*list*); usato da solo mostra l'elenco dei files nella directory corrente

```
$  ls
DXclock                 dich.pdf                oolang.html
DXterm                  dich.ps                 regis.html
Desktop                 higz_windows.dat        sat.gif
GNUstep                 icons                   tex
bin                     mail
copy                    nsmail
$
```

mentre accompagnato dal nome (assoluto o relativo) di una directory mostra i files contenuti nella stessa

```
$  ls  /bin
arch            dd              gzip        netconf         sleep
ash             df              hostname    netstat         sort
ash.static      dmesg           igawk       nice            stty
.
.
.
consolechars    gawk            mktemp      rview           vi
cp              gawk-3.0.4      more        sed             view
cpio            grep            mount       setserial       vimtutor
csh             gtar            mt          sfxload         ypdomainname
date            gunzip          mv          sh              zcat
```

Se oltre al nome di un file si vogliono avere tutte le informazioni ad esso relative, si può aggiungere l'opzione *-l* (per *long*) a *ls*

```
$  ls  -l  /bin
total 5484
-rwxr-xr-x    1 root     root         2612 Mar  7  2000 arch
-rwxr-xr-x    1 root     root        60592 Feb  3  2000 ash
-rwxr-xr-x    1 root     root       263064 Feb  3  2000 ash.static
.
```

```
.
.
-rwxr-xr-x    1 root     root          362 Mar  7  2000 vimtutor
-rwxr-xr-x    3 root     root        46384 Feb 15  2000 zcat
```

la prima colonna indica le protezioni: il primo "–" indica che si tratta di un file normale (d indica una directory, l un link simbolico, e così via), cui seguono i nove bit delle protezioni (ad esempio, il file `arch` può essere letto, modificato ed eseguito dall'utente, solo letto ed eseguito dal gruppo e da tutti gli altri; si veda il par. 3.8). Il numero che segue indica il numero di link, ovvero in quante directory compare un riferimento all'i–node di quel file (solitamente è 1 per i files normali e 2 per le directory). Ci sono quindi il nome del proprietario (*root* nell'esempio, trattandosi di files di sistema: root infatti rappresenta l'Amministratore della macchina) e il nome del gruppo (che nello specifico coincide col nome dell'utente, ma è un caso particolare per root). Vi è quindi la dimensione in byte del file, la data di creazione (che se è più recente di 6 mesi è indicata come giorno, mese, ora e minuti, mentre oltre i 6 mesi è indicata come giorno, mese e anno; si tratta comunque solo dell'indicazione di *ls*, giacché nell'i–node la data è conservata completa), e il nome del file. All'inizio compare pure la scritta `total` che indica lo spazio totale (in kB) occupato da tutti i files nella directory in esame.

Da questo esempio si può trarre una osservazione generale, che vale per la stragrande maggioranza dei comandi Unix: ogni comando ha la sintassi generale

$$comando \qquad opzioni\ (facoltative) \qquad argomenti$$

nell'esempio

$$\underbrace{ls}_{comando} \quad \underbrace{-l}_{opzione} \quad \underbrace{/bin}_{argomenti}$$

Quasi sempre le opzioni sono precedute da un segno "–" e spesso possono essere ragruppate fra loro, cioè *ls -l -a* è equivalente a *ls -la*.

Con *ls* da solo però non si vedono tutti i files presenti: vengono omessi i cosiddetti *files nascosti* (*hidden files*). Solitamente (anche se non sempre) si tratta di files o directory di configurazione che normalmente non si ha interesse a vedere; essi possiedono tutti la caratteristica di avere un nome che comincia con un ".". Per vedere anche questi occorre aggiungere l'opzione -a (*all*) a *ls*

```
$  ls -a
.                 .bash_profile       .mailcap          .user52.rdb
..                .bashrc             .mc               .xsession-errors
.ICEauthority     .emacs              .mime.types       higz_windows.dat
.Xauthority       .enlightenment      .netscape         nsmail
.acrorc           .gnome              .screenrc         office52
.bash_history     .gnome-desktop      .ssh
.bash_logout      .kde                .sversionrc
```

Da notare che "." indica la directory corrente, mentre ".." la directory immediatamente superiore. Ecco quindi che per salire di un livello si può usare *cd ..*

```
$  pwd
/home/users/auser/data
$  cd ..
$  pwd
/home/users/auser
```

Si cita infine il comando *df* che mostra i dischi e le partizioni montate, la loro capacità complessiva, lo spazio occupato e quello ancora libero e la directory su cui sono montate (cioè il mount point, si veda il par. 3.8)

```
$  df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda2               995147    248657    695082  26% /
/dev/hda4              1895849   1324496    473357  74% /usr
/dev/hda1              1124296    480172    644124  43% /dos
```

(su alcuni sistemi la capacità è data in blocchi da 512 byte invece che da 1 Kb; si può allora usare l'opzione *-k* per forzare la lettura in Kb).

Esame e alterazione dei files

Per esaminare il contenuto di un file si può usare il comando *cat* (per *catenate*, perché con questo comando è possibile concatenare più files in uno): per esempio

```
$  cat .acrorc
*PrefsVersion:    3
*OpenDialogAtStartup:      false
*ShowSplashAtStartup:      true
.
.
.
*SendFarEastFonts:            true
*SuppressCSA:   false
*PaperSize:          6
```

Lo scorrimento è però veloce, e se il file è molto lungo non si farà a tempo a leggere le righe iniziali prima che escano dallo schermo. Si può allora fare ricorso al comando *more*

```
$  more .acrorc
*PrefsVersion:    3
*OpenDialogAtStartup:      false
*ShowSplashAtStartup:      true
.
.
.
*NoteColor:          #ffffffff0000
*MaxThreadZoom:  8
*EnablePageCache:              false
--More--(25%)   Spazio
*EnablePageCache:              false
*FullScreenColor:              #000000000000
*FullScrolling:  false
.
.
.
```

esso mostra una schermata alla volta e attende la pressione della barra spaziatrice per continuare, mentre se si preme il tasto *b* è possibile tornare alla schermata precedente; con *q* invece si può annullare la visualizzazione e tornare al prompt (*more* ha altri comandi utili, ma questi sono quelli più comunemente usati).

Questi comandi consentono di esaminare il contenuto di un file ma non di modificarlo. Per fare ciò è necessario usare un *editor*. In Unix ne esistono molti, ma i due di gran lunga più diffusi sono *vi* e *emacs*. Il primo è esclusivamente un editor di schermo, cioè la manipolazione del file avviene nello stesso schermo in cui si inseriscono i comandi della shell, il secondo invece apre una finestra a parte se si sta lavorando con una interfaccia grafica, mentre usa la stessa schermata se si lavora con una interfaccia di schermo. Ciascuno ha pregi e difetti: probabilmente *vi* è più semplice da usare per compiti non molto complessi, mentre *emacs* ha più funzionalità; quest'ultimo poi in ambiente grafico presenta dei menu a tendina per le operazioni più comuni, e contiene anche un tutore interno che illustra con esempi i principali comandi da tastiera. Entrambi possono essere invocati da soli o seguiti dal nome del file da modificare

```
$  emacs
$  vi dati.txt
```

Fra gli editor più comuni che possono essere trovati nei sistemi Unix ci sono pure *nedit* e *xedit.*

Copie e spostamenti dei files

Per copiare un file esiste il comando *cp* (*copy*) che crea una nuova copia del file: la sintassi generale è *cp vecchiofile nuovofile* per copiarlo con un'altro nome, oppure *cp vecchiofile nuovadir* per copiarlo in una nuova directory:

$ *cp prova.dat nuovo.dat*
(copia `nuovo.dat` *nella stessa area)*
$ *cp prova.dat /tmp/auser*
(copia `prova.dat` *in* `/tmp/auser`)
$ *cp prova.dat /tmp/auser/nuovo.dat*
(copia `prova.dat` *in* `/tmp/auser` *chiamandolo* `nuovo.dat`*)*

Con il comando *mv* (*move*) è invece possibile spostare un file da una locazione ad un'altra ovvero cambiargli nome

$ *mv prova.dat nuovo.dat*
(muove `prova.dat` *in* `nuovo.dat` *nella stessa area)*
$ *mv prova.dat /tmp/auser*
(muove `prova.dat` *in* `/tmp/auser`)
$ *mv prova.dat /tmp/auser/nuovo.dat*
(muove `prova.dat` *in* `/tmp/auser` *chiamandolo* `nuovo.dat`*)*

in ogni caso non viene creata una nuova copia sul disco, ma viene solo spostata quella esistente.

Compilazione dei programmi in C

Il compilatore C viene invocato col comando *cc* seguito dal nome del file sorgente

$ *cc prova.c*

Il programma eseguibile viene sempre chiamato `a.out` qualunque sia il nome del sorgente; per assegnare un nome diverso (altamente consigliato) si usa l'opzione *-o* (per *output*)

```
$ cc prova.c -o prova
$ ls
prova    prova.c
```

Se il programma richiede una o più librerie, ognuna di esse va segnalata al compilatore preceduta dall'opzione *-l*: così ad esempio il comando

$ *cc prova.c -o prova -l myprogs*

compila il programma `prova.c` chiamando l'eseguibile `prova` e includendovi i moduli contenuti nel file di libreria `libmyprogs.a`. Va ricordato che per convensione i files di libreria hanno estensione `.a` e il loro nome è preceduto dal prefisso `lib`. Il compilatore cerca i files di libreria nella directory corrente e in alcune directory di sistema; se la libreria si trova in una directory non standard, si può segnalare quest'ultima al compilatore con l'opzione *-L*: pertanto se `libmyprogs.a` si trova ad esempio in `/usr/local/lib` si potrà scrivere

$ *cc prova.c -o prova -L /usr/local/lib -l myprogs*

Altre informazioni sul sistema

Lo Unix è un sistema multiprogrammato: usando il comando *ps* è possibile vedere tutti i processi dell'utente

```
$ ps
  PID TTY          TIME CMD
  583 pts/0    00:00:00 bash
  588 pts/0    00:00:34 emacs
 1464 pts/0    00:00:00 ps
$
```

esso fornisce l'identificatore del processo, il terminale, il tempo per cui il processo è rimasto attivo e il programma che esso sta eseguendo. Questo comando ha molte opzioni che consentono di esaminare tutti i processi in corso e di ricavarne utili informazioni.

Il comando *date* fornisce la data e l'ora correnti del sistema

```
$ date
Mon Mar  5 09:17:05 CET 2001
```

Informazioni in linea

E' possibile ottenere informazioni e aiuto in linea: per ogni comando infatti esiste una serie di *pagine di manuale* (*manual pages*) con una spiegazione sommaria e l'elenco delle opzioni accettate, spesso corredata da alcuni esempi. Per invocare la pagine di manuale di un comando si usa *man comando*, ad esempio

$ *man ls*

```
LS(1)                          FSF                            LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]...  [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by
       default).  Sort entries alphabetically if none of -cftuSUX
       nor --sort.
        -a, --all
              do not hide entries starting with .
        -A, --almost-all
              do not list implied .  and ..
.
.
.
```

Bibliografia

L. Greenfield, *The Linux Users' Guide*

Indice