Capitolo 3: Il sistema operativo

3.1 Livelli di funzionalitá di un calcolatore

La figura 3.1 rappresenta la gerarchia dei termini che formano i livelli di funzionalitá del PC. Se con hardware indichiamo tutte le parti fisiche dell'eleboratore notiamo che questo livello contiene una parte detta firmware che individua la parte, normalmente registrata in una memoria ROM, dove sono presenti microistruzioni necessarie per la fase di avvio (bootstrap) e per il caricamento del sistema operativo.

Il livello piú basso é il **linguaggio macchina** (sequenza di numeri binari) che é l'unico linguaggio che riesce la CPU riesce a comprendere.

Non é molto comodo scrivere un programma utilizzando questo linguaggio; il divario tra un linguaggio macchina e un linguaggio piú evoluto é colmato dal sistema operativo (OS Operating System).

Un sistema operativo si preoccupa di:

- gestire l'interfaccia utente cioé di interpretare ed eseguire una serie di comandi comunicati dall'utente tramite tastiera o mouse. Ció permette di manipolare insiemi di dati (files) e di accedere alle periferiche senza avere una conoscenza approfondita dell'hardware del PC.
- permettere l'utilizzo contemporaneo della CPU da parte di piú programmi.
- gestire l'utilizzo della memoria centrale.

Probabilmente molti di voi conoscono i sistemi operativi commerciali piú diffusi (DOS, Windows95, 98, NT, 2000, ME etc.., MAC), durante il corso utilizzeremo un sistema operativo completemente gratuito chiamato Linux, che deriva da un sistema operativo creato per workstation (Unix) e adattatato all'architettura dei PC.

3.2 Linux

3.2.1 Un po' di storia

Linux è una versione per PC del sistema operativo (molto potente) Unix studiato per la gestione delle workstation utilizzate come server (ovvero macchine potenti in grado di gestire un centro calcolo con molti utenti e reti locali).

Il sistema UNIX sviluppato dalla AT&T Bell Laboratories nacque essenzialmente come reazione ai sistemi operativi di grandi dimensioni, complessi e poco flessibili, disponibili alla fine degli anni sessanta. Il principale obiettivo di progettazione fu quello di ottenere un sistema operativo d'elevata potenzialità.

Originariamente (anni '70) il sistema operativo fu sviluppato da un gruppo di esperti di computer come uno strumento di sviluppo per uso personale, ignorando cosí le esigenze dei principianti a vantaggio della velocitá e della precisione. A causa delle leggi antitrust il software non poteva essere venduto e prese la strada delle Universitá Americane che a loro volta iniziarono a sviluppare varie versioni di Unix facendolo maturare velocemente ma creando anche un po' di disordine e confusione. In questo modo iniziarono a formarsi una casta di persone che una volta laureate e pronte per il mondo del lavoro iniziarono a divulgare e far conoscere Unix nelle aziende. Pian piano Unix potè diventare un prodotto commerciale e si iniziarono a vedere sul mercato varie versioni proprietarie e quindi a pagamento. Linux è nato come estensione del Minix (sistema operativo Unix per processori i86 sviluppato ad uso didattico dal professor Andrew

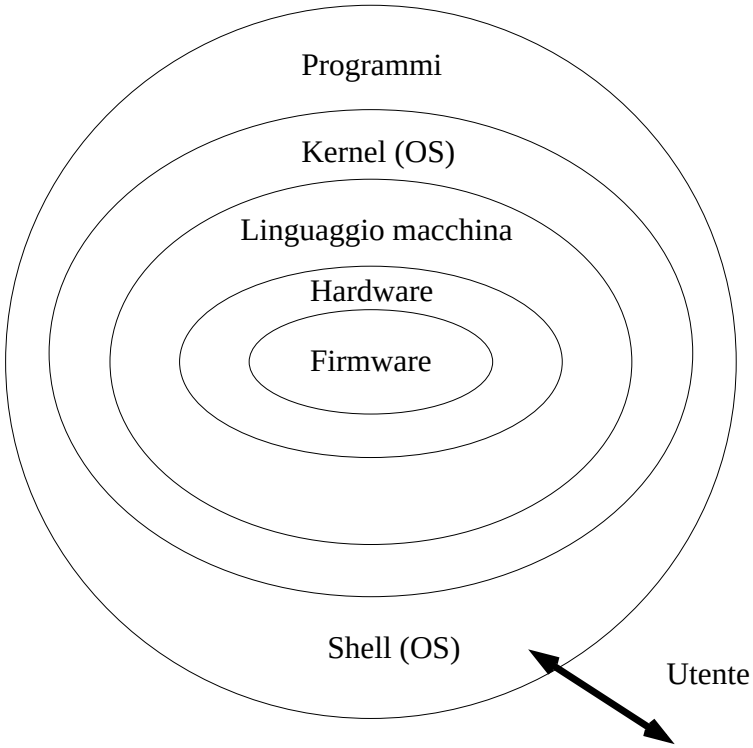

Figura 3.1: Il livelli di funzionalità di un calcolatore

S. Tanenbaum) e come clone di Unix, dallo studente finlandese dell' Universitá di Helsinki Linus Beneditc Torvalds, per un progetto universitario sullo studio delle funzionalitá multitasking dei microprocessori i386.

Linux viene mantenuto da un gruppo di programmatori sparsi per tutto il mondo e la sua flessibilitá e la sua potenza sono accompagnati da un vantaggio che non deve essere trascurato. Linux è completamente gratuito cioé non "si è obbligati" a pagare per poter utilizzare questo sistema operativo. In pratica esiste una licenza GNU (sigla ricorsiva GNU's not Unix) che assicura che Linux rimanga gratuito ma anche standard.

Uno dei vantaggi principali di Linux è che in esso sono forniti tutti gli strumenti per il calcolo scientifico, come per esempio: editori di testo molto complessi, tutti i linguaggi di programmazione, librerie grafiche etc..

Non è da trascurare il fatto che è un sistema multiutente e multitasking, ovvero piú utenti possono utilizzare lo stesso computer (da terminali differenti) e richiedere al sistema di eseguire piú operazioni (task) contemporaneamente.

Recentemente le interfacce grafiche per l'utilizzo del sistema operativo sono si fatte sempre piú elaborate e "simili" a quelle dei sistemi commerciali.

Per la cronaca, dopo sette anni dalla nascita di Linux, Linus Torvalds si é laureato e dopo aver collaborato per diversi anni come ricercatore all'interno dell'Universitá di Helsinki si é trasferito in America dove continua, anche se non come attivitá primaria, a dirigere lo sviluppo del sistema operativo e a lavorare su una versione real-time di Linux.

3.2.2 La struttura di Linux: il kernel e la shell

Il sistema operativo è costituito da un sistema di base e da un certo numero di estensioni. Il sistema di base comprende il kernel, le shell e le utility di base. Le estensioni sono ad esempio le funzionalitá grafiche, le librerie , i compilatori. Il kernel di Linux, che consiste nell'interfaccia tra l'hardware e i processi, gestisce l'accesso alla CPU e la memoria, e si occupa, fra le altre cose, del controllo e della gestione di ciascuna periferica (device). Rappresenta, quindi, il nocciolo dell'intero sistema operativo e idealmente puó assumersi come una sorta di astrazione della macchina fisica essendo posto al livello software piú basso.

Per quanto riguarda le versioni del kernel c'e' da dire che esse utilizzano uno specifico schema per la loro numerazione : le versioni V.x.y con x pari rappresentano le versioni stabili, mentre al contrario le versioni con x dispari sono le beta-release (ancora in prova) utilizzate normalmente dagli sviluppatori e possono essere instabili o generare effetti collaterali non conosciuti: mano a mano che si risolvono i bug (bachi!) il numero y viene incrementato.

Bisogna stare attenti a non confondere la versione del kernel con la distribuzione che contiene anch'essa un numero. Esistono infatti diversi distributori di Linux (cambia praticamente solo l'aspetto grafico e alcune utilities) RedHat, Mandrake, Slackware, Suse, Debian, quindi posso avere per esempio la Mandrake 8.2 che fornisce il kernel 2.4.3-20mdk (V=2,x=4,y=3).

La shell è il collegamento tra l'utente e il kernel. Mentre il kernel è definito, per una data installazione, una volta per tutte, esistono diverse possibili realizzazioni di SHELL per lo stesso computer, che possono essere presenti simultaneamente nello stesso sistema ed eventualmente essere richiamate o abbandonate dall'utente nel corso di una sessione interattiva.
Per ogni utente viene definita una cosidetta SHELL di login, ossia che è attiva all'atto del collegamento; nel nostro caso la SHELL di login è la "**tcsh**".
È importante osservare che le SHELL Linux distinguono le lettere maiuscole da quelle minuscole; in questo senso si dice che Linux è "case sensitive".

I comandi Linux sono terminati dal tasto **Enter** che serve a passare il comando stesso al sistema operativo per la sua esecuzione. Essi possono contenere delle opzioni e/o dei parametri, per i quali occorre rispettare la stessa distinzione tra lettere minuscole e maiuscole.
Per richiamare comandi eseguiti in precedenza si possono utilizzare i tasti **Up** e **Down** (↑ e ↓).
Il comando `history n` mostra su terminale gli ultimi n comandi eseguiti.

3.3 Struttura dei files e delle directories

Dal punto di vista logico un file è un contenitore di informazioni memorizzate in modo permanente (cioè che non vengono perse quando il computer viene spento). Dal punto di vista fisico un file è una porzione di un disco magnetico sulla quale si può scrivere, leggere e cancellare.

Ogni file possiede un nome, che ne consente l' identificazione. Inoltre, per consentire una migliore gestione dello spazio disponibile, i files sono, dal punto di vista logico, racchiusi in contenitori che si chiamano "directories". Ogni directory può contenere files o altre directories, di modo che l' intero sistema (detto "filesystem") risulta organizzato ad albero (Fig. 3.2). Il nome completo di un file è dato dalla serie completa dei nomi delle directories in cui è contenuto a partire dalla radice dell' albero oltre che dal suo nome proprio. La radice si chiama **/**, e lo stesso carattere **/** divide i nomi delle directories ed il nome dell' ultima directory dal nome proprio del file. Ad esempio il nome del file **prova.c** nella directory **labc** sarà

/usr/users/cognome1/labc/prova.c

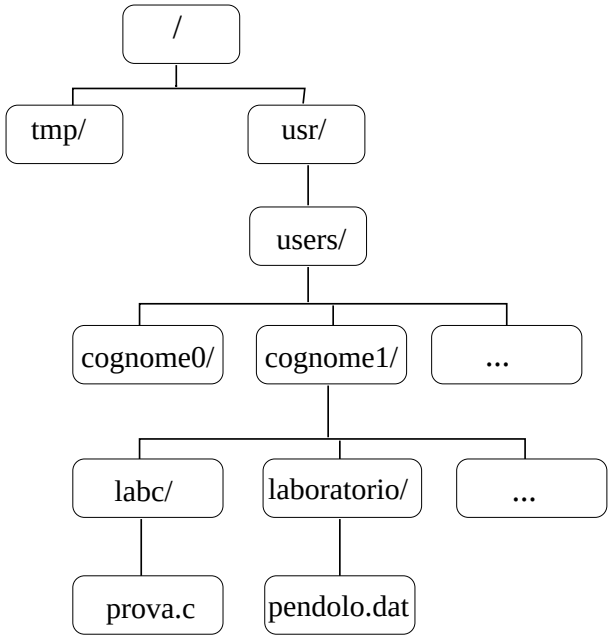

Figura 3.2: Esempio di struttura ad albero di directories

Ovviamente è piuttosto scomodo utilizzare i nomi completi dei files, visto che di solito sono lunghi; di conseguenza Linux fornisce alcuni strumenti per abbreviare i nomi. Il più importante si chiama "working directory" (WD): si tratta di una particolare directory che viene usata come punto di partenza per espandere i nomi di files che non cominciano per **/**. Ad esempio, se la WD è **/usr/users/cognome1** il nome del file precedente può essere abbreviato in

labc/prova.c

Per sapere quale è la WD si usa il comando **pwd**; per cambiare la WD si usa il comando **cd**, che accetta come parametro il nome di una directory (con o senza **/** iniziale a seconda che si voglia partire dalla radice o dalla WD).

Il nome della WD può essere abbreviato con il simbolo **.** (punto), mentre per la directory genitrice della WD si può usare il simbolo **..** (due punti). Ad esempio, se siamo nella directory **labc**, potremmo accedere al file **pendolo.dat** nella directory **laboratorio**,utilizzando il nome

../laboratorio/pendolo.dat

in quanto, se la WD è **labc**, **..** è un sinonimo di

/usr/users/cognome1/

All'inizio di una sessione la WD è una directory specifica per ciascun utente (il cui nome coincide con lo username); questa directory viene detta "home directory" (HD) e il suo nome può essere abbreviato con il simbolo ~.

Ogni utente può creare nuove directories all' interno della HD utilizzando il comando **mkdir**.

La quantità di dati che ogni utente può scrivere nella propria HD e nelle directories che essa contiene è limitato. Il comando **quota** consente di verificare la situazione del proprio account in termini di spazio libero (i dati sono espressi in kbytes). Nel caso lo spazio disponibile nella HD non fosse sufficiente, è possibile utilizzare un' area temporanea per salvare files di grandi dimensioni; l' area temporanea si chiama **/tmp**. È evidente che devono essere messi in tale area solo files temporanei o facilmente ricreabili; ad esempio i files eseguibili ottenuti dalla compilazione di un sorgente C possono essere scritti in **/tmp**.

3.4 Controllo dei processi

Come già detto Linux è un sistema multi-task ed ogni SHELL può eseguire diversi processi contemporaneamente. Il comando `ps` mostra tutti i processi che sono in esecuzione nella SHELL. Questo comando è utile per trovare l' identificatore (PID) di un processo che non risponde e che si vuole fermare. L' arresto di processo si esegue con il comando `kill PID`.
Linux permette di sospendere l' esecuzione di un programma e riattivarla in seguito.
Quando un programma è "lanciato" dalla linea di comando digitando il suo nome, si dice che tale processo è eseguito in "foreground" (il terminale resta bloccato fino al termine del programma). Un programma in "foreground" che non risponde può essere fermato bruscamente con Ctrl-C, o sospeso con Ctrl-Z.
Se un stesso programma è "lanciato" aggiungendo dopo il nome il carattere &, si dice che il programma è avviato in "background" (il terminale è libero di ricevere comandi).
Il comando `jobs` mostra la lista dei programmi in "background". Ogni programma è preceduto da un numero (`n`). Per fermare un programma in "background" si esegue il comando `kill %n`. Il comando `fg %n` (`bg %n`) manda in "foreground" ("background") il programma corrispondente. Se `n` non è specificato `fg` e `bg` agiscono sul programma corrente (quello con il + nella lista stampata da `jobs`).

3.5 Nomi dei files e delle directories

Come abbiamo giá detto il SO Linux é "Case sensitive" quindi un file o una directory con il nome **Pippo** é distinta da quello/a con il nome **pippo**.

I nomi dei files e delle directories non possono contenere nessuno dei seguenti caratteri: spazio, ?, * ,! ,apici, accenti e virgolette varie, <, >, |, il carattere separatore '/', '%', '$', '; parentesi varie. Questi sono caratteri che hanno speciali significati per la shell. Anche far cominciare il nome di un file con il carattere - non é una buona idea, in quanto usato per indicare le opzioni di un comando.

3.6 Comandi per la gestione di files e directories

La maggior parte dei comandi del SO agisce su files: di conseguenza è molto importante imparare bene a scrivere i nomi dei files ed a manipolare le directories.
La Tab. 3.1 contiene è una lista non esaustiva dei comandi per la manipolazione di files in Linux, le opzioni riportate sono solo quelli più utilizzate: *nome* indica un generico nome di file o di directory, mentre *fil* e *dir* si riferiscono in modo specifico a un file e a una directory. Le opzioni si possono combinare; se, per esempio, si vuole eseguire il comando `ls` con l' opzione `-a` e `-l` si scriverà `ls -la`.
Per una descrizione più completa si rimanda al manuale online (vedere Par. 3.12).

comm	opt		
pwd			Stampa su schermo il nome della WD (Working Directory)
echo		*textline*	Mostra la linea di testo *textline*
cd		*dir*	Cambia la WD in dir
ls		*nome*	Lista i files contenuti nella dir specificata
	-l		stampa infomazioni aggiuntive sui files (data, protezioni)
	-a		lista tutti i files (compresi quelli del tipo .nome)
	-F		aggiunge un classificatore tra i simboli (*/=@—) a ciascun file
	-t		ordina i files per data dell' ultima modifica
mkdir		*dir*	Crea *dir*
rmdir		*dir*	Cancella *dir*
mv		*nome1 nome2*	Ridenomina *nome1* in *nome2*
cp		*nome1 nome2*	Copia *nome1* in *nome2*
	-r		copia il contenuto della dir *nome1* nella dir *nome2*
rm		*fil*	Cancella *fil*
	-i		chiede conferma prima di cancellare ciascun file
cat		*fil*	Stampa su schermo il contenuto di *fil* (accetta più files)
less		*fil*	Impagina il contenuto di *fil* su schermate
			(spazio → schermata successiva, b → schermata precedente)
tail	-n *N*	*fil*	Stampa su schermo le ultime *N* linee di *fil*
head	-n *N*	*fil*	Stampa su schermo le prime *N* linee di *fil*
grep		*string fil*	Cerca la stringa *string* nel file *fil*
	-i		non fa differenza tra lettere maiuscole e minuscole (in *string*)
find	-name	*fil*	Cerca un file in WD e nelle sue sotto-directories
diff		*fil1 fil2*	Stampa su schermo le linee di testo diverso tra due files
ln	-s	*fil link-name*	Crea un equivalenza simbolica tra *link-name* e *fil*
			(tutte le operazioni eseguite su *link-name* sono applicate a fil)
	-f		se *link-name* esista già lo cancella
lpr	-P *printer*	*fil*	Manda in stampa sulla stampante *printer* il file *fil*

Tabella 3.1: Breve elenco di alcuni comandi Linux utili per la gestione di files

3.6.1 Utilizzo di wildcard

Può essere utile, in alcuni casi, lasciare le lettere non definite nei nomi dei files (wildcards) (ad esempio se si vogliono eseguire comandi su più files aventi una stringa di caratteri comune nel nome).
Le wildcards consentite sono:

*	sostituisce qualsiasi stringa di caratteri (di qualsiasi lunghezza)
?	sostituisce un singolo carattere
[03]	sostituisce il carattere 0 o 3
[0-3]	sostituisce i caratteri corrispondenti ai numeri da 0 a 3 (per i caratteri vale l' ordine alfabetico)

Es.:
Se in una directory sono contenuti i files: test00.f test1.f test2.f test3.f, il comando `ls test*.f` li listerà tutti, `ls test?.f` listerà solo gli ultimi tre e il comando `ls test[12].f` listerà il secondo e il terzo.

3.6.2 Pipe e redirezione

In Linux i comandi possono essere eseguiti in sequenza. Per passare l' output di un comando come input al seguente si utilizza il carattere | (chiamato pipe). Vediamo un esempio:

```
ls | grep ps | less
```

questo comando lista il contenuto dei files nelle directory (`ls`), cerca quelli che contengono la stringa ps nel nome (`grep`) e li passa a `less` che li visualizza su schermo.
I simboli `<` e `>` permettono di ridirigere lo standard input (tastiera) e lo standard output (terminale).
Se scriviamo:

```
ls > listato
```

il listato della directory invece di comparire sul terminale viene scritto sul file *listato* (se non esiste il file viene creato, se esiste sovrascritto).
Il comando

```
ls >> listato
```

è simile al precedente salvo che, ora, il risultato di `ls` è scritto in coda al file *listato* (se non esiste il file viene creato).

3.6.3 Protezioni

Le protezioni definiscono come i diversi utenti possono accedere a files e directories.
Le protezione per un file/directory possono essere controllate con il comando `ls -l` che fornisce (per una directory) un risultato di questo tipo:

```
drwxr-xr-x   2 smith                1024 Oct  7 20:01 Work
```

La legenda della prima serie di caratteri (che contiene le protezioni) è data dalla seguente tabella

```
Type  User Group  Other
 *    ***    ***    ***
 d    rwx    rwx    rwx
 l    ---    ---    ---
 -
```

Il primo carattere definisce il tipo di file (d=directory,l=link simbolico,−= file), gli altri regolano i privilegi di accesso ai files (r=lettura,w=scrittura,x=esecuzione) da parte di diversi gruppi di persone.
Lo User è l' utente che si è collegato, Group il suo gruppo di appartenenza, Other tutte le altre persone che possono avere accesso a quella macchina.
Per modificare le protezioni si utilizza il comando `chmod`.
Es.:
`chmod g+x` *fil*: aggiunge a Group il diritto di eseguire *fil*
`chmod o-x` *fil*: toglie a Group il diritto di leggere *fil*

3.7 Il text editor

Il text editor è uno strumento che serve a creare files; ogni tipo di file "leggibile" (un testo, il sorgente di un programma, un file di dati) può essere creato con un text editor. Esistono molti tipi di editor, e la scelta dipende essenzialmente dai gusti personali. Quello che vi consigliamo si chiama **emacs**. Per attivarlo è sifficiente dare il comando

emacs nome-fil &

o semplicemente

emacs &

Se la variabile DISPLAY (Par. 3.13) è correttamente definita, verrà creata sul vostro X-terminal una nuova finestra per l' editor; **emacs** possiede un numero enorme di comandi che consentono funzioni di editing molto sofisticate, ma la descrizione di queste possibilità esula dagli scopi di queste note. Il menu "Help" contiene tutte le informazioni che servono per usare al meglio **emacs**. Basterà qui osservare che **emacs** è un editor full-screen, e che quindi è possibile muoversi all' interno del file usando i tasti con le frecce; inoltre la maggior parte dei comandi possono essere attivati usando il mouse e la "menu-bar" che appare nella parte alta della finestra. È possibile aprire più files contemporaneamente, utilizzando l' opzione "Open File" nel menu "Files" e il menu "Buffers" per scegliere tra i diversi files. Per salvare un file modificato si usa il comando "Save Buffer" nel menu "Files".
Siccome disponete, per le esercitazioni, di terminali che consentono di lavorare su più finestre, è conveniente evitare di aprire e chiudre l' editor in continuazione. All' inizio della seduta potrete aprire **emacs** utilizzando la "&" a fine riga in modo da creare una finestra indipendente per l' editor e lasciare il vostro terminale libero di eseguire i comandi di compilazione ed esecuzione. Dopo avere eseguito una modifica al vostro file, salvatelo senza uscire dall' editor con "Save Buffer" nel menu "Files", compilate ed eseguite utilizzando il terminale e tornate a modificare il file nella finestra dell' editor, che dovrà essere chiusa solo alla fine del lavoro.

3.8 Compilazione ed esecuzione di un programma

In questo paragrafo daremo una breve descrizione dei comandi necessari per compilare un programma allo scopo di realizzare il primo semplice programma in C. Un' introduzione più completa sarà fornita in Par. 5.2.

Il comando di compilazione permette di tradurre un programma scritto in linguaggio di "alto" livello (in linguaggio C nel nostro caso) in un codice eseguibile dalla macchina. La compilazione prevede in genere due fasi: la compilazione vera e propria che trasforma il file sorgente (.c) in file oggetto (.o) e la fase di "link" in cui si produce il file eseguibile includendo, se necessario, files di libreria o altri files oggetto.
Il comando:

```
cc -o file file.c
```

compila il sorgente `file.c` e crea l' eseguibile `file`.
Questo comando crea in un solo passo l' eseguibile ed è equivalente a

```
cc -c -o file.o file.c     (compilazione)
cc    -o file   file.o     (link)
```

Il file eseguibile può essere eseguito semplicemente digitando il suo nome completo:

```
./file
```

3.9 Il debugger simbolico

Il debugger simbolico è uno strumento software che consente di eseguire un programma una istruzione alla volta e che permette di esaminare il contenuto delle variabili (sia locali che globali) in una qualunque fase dell' esecuzione. È quindi uno strumento utilissimo per capire le cause di malfunzionamento di un programma. Esistono sul mercato vari tipi di debugger: quello che vi consigliamo si chiama **gdb**.

Esistono due modi per utilizzare un debugger; il primo modo è "post–mortem", nel senso che utilizza il file chiamato `core` che ogni programma genera in caso di errore. In questo caso i comandi da usare sono:

```
gdb <nome programma> core
where
```

Il comando `where` del debugger vi informarà di quale linea del programma ha causato l' errore. Per uscire dal debugger utilizzate il comando `quit`.

Il secondo utilizzo del debugger è quello di eseguire un programma passo–passo. In questo caso si comincia con

```
gdb <nome programma>
```

Si deve quindi predisporre almeno un "breakpoint", ovvero una posizione del programma alla quale l' esecuzione deve sospendersi; un breakpoint può essere specificato in due modi:

```
break <routine>   ! Sospende l' esecuzione all' ingresso nella routine
```

oppure

```
break <numero linea>  ! Sospende l' esecuzione alla linea specificata
```

Il numero di linea da specificare è quello del programma sorgente.

Dichiarati i breakpoints si può cominciare l' esecuzione con il comando `run`. Quando l' esecuzione si interrompe si può:

- riprendere fino al successivo break con il comando `cont`;
- eseguire solo la linea successiva senza entrare in eventuali routines chiamate con il comando `next`;
- eseguire solo la linea successiva entrando in eventuali routines chiamate con il comando `step`;
- proseguire fino all' uscita dalla routine corrente con il comando `return`;
- leggere il file sorgente nel punto corrispondente alla linea successiva a quella appena eseguita con il comando `list`;
- esaminare il contenuto di una variabile visibile nella routine corrente con il comando `print <nome_variabile>`;
- modificare il contenuto di una variabile con il comando `set <nome_variabile> = <valore>`;
- creare nuovi breakpoints.

Le componenti dei vettori possono essere esaminate individualmente, usando le parentesi quadre (per i programmi scritti in C).

L' utilizzo di questi comandi, e dei molti altri disponibili per i quali potete consultare il manuale (vedi Par. 3.12), consente di trovare le cause di malfunzionamenti e può aiutare a comprendere meglio il modo di operare di alcune istruzioni dei linguaggi di programmazione.

Il debugger **gdb** possiede una versione grafica cui potete accedere con il comando **xxgdb**. In questa versione il file sorgente è sempre visibile ed è possibile definire un breakpoint semplicemente "cliccando" a fianco di ogni linea.

3.10 Procedure di comandi

Accade sovente di dover usare ripetutamente sequenze di comandi complesse. Per consentire un risparmio di tempo ed un incremento di flessibilità gli interpretatori dei comandi (shell) sono dotati di un certo grado di programmabilità: è cioè possibile scrivere delle sequenze di comandi in un file (che viene detto "script") ed eseguirle scrivendo un solo comando. Inoltre all' interno di uno script si possono utilizzare variabili e strutture complesse tipo if-then-else, do-while e così via. Un qualsiasi manuale di Linux riporta in dettaglio le possibilità di programmazione incluse nei diversi tipi di shell (quella che noi utilizziamo si chiama `csh`,"C shell"[1]). La scelta della "C shell" è dettata dal fatto che, come dice il nome, la sua sintassi è molto simile a quella del C.
Nel seguito analizzeremo solo alcuni elementi sintattici di base che possono servire per scrivere scripts molto semplici.

[1] la shell di login `tcsh` è semplicemente la versione "interattiva" della `csh`

3.10.1 Uso delle variabili

Nella C shell possono essere definiti variabili e vettori di tipo stringa. Esistono variabili locali e variabili globali: le prime esistono solo all' interno della procedura che le crea, le seconde restano attive nell' arco di tutta la sessione. Le variabili locali vengono dichiarate con il comando **set**, mentre quelle globali con il comando **setenv**; ad esempio:

`set a1 = val_1`	Crea la variabile locale a1 e le assegna il valore val_1
`setenv g1 val_2`	Crea la variabile globale g1 e le assegna il valore val_2
`set vect = (v1 v2 v3)`	Crea il vettore a tre componenti vect ed assegna i valori v1, v2 e v3 alle tre componenti

Il contenuto della variabile `a1` è dato dal simbolo `$a1`, il valore dell' n-esima componente del vettore `vect` dal simbolo `$vect[n]`, mentre il numero di componenti del vettore `vect` è dato da `$#vect`.
È possibile assegnare ad una variabile l' output di un comando, includendo il comando medesimo tra apici rovesciati (‘); ad esempio

```
set files = (‘ls‘)
```

crea un vettore di nome `files` che contiene i nomi dei file presenti nella WD, dati come output dal comando **ls**.
Alcune variabili globali sono automaticamente definite dal sistema nella fase di login. Per vedere quali sono basta eseguire il comando `setenv` senza argomenti.
Una importante variabile globale è *PATH* che contiene la lista delle directories in cui il sistema cerca un file eseguibile se questo non è specificato con il nome completo.

3.10.2 Realizzazione ed esecuzione di una procedura di comandi

Una procedura di comandi può essere scritta con un text editor. La prima riga deve contenere la specificazione della shell che eseguirà i comandi. Nel caso della C shell la prima riga sarà

```
#!/usr/bin/csh
```

Le linee successive conterranno la sequenza di comandi che si desidera eseguire. Le righe che cominciano com `#` sono considerate commenti e non vengono eseguite. Una volta preparato il file è necessario renderlo eseguibile con il comando

chmod `u+x nome-fil`

Per eseguirlo basta quindi scriverne il nome completo, o solo il nome proprio nel caso lo script sia in una directory contenuta nel `PATH`. Il nome dello script può essere seguito da uno o più parametri. Questi verranno passati allo script come un vettore locale di nome `argv`. Di seguito trovate il listato di uno script che esegue compilazione e link del sorgente specificato come primo argomento ed esegue il programma così ottenuto. Se il secondo parametro è `-c` viene eseguita la sola compilazione, se è `-r` il programma viene solo eseguito.

```
#!/usr/bin/csh
if ($#argv == 1) then
  cc -o /tmp/$argv[1] argv[1] fun1.c ‘cernlib graflib‘
```

```
  chmod u+x /tmp/$argv[1]
  /tmp/$argv[1]
else
  switch ($argv[2])
    case -c:
    cc -o /tmp/$argv[1] argv[1] fun1.c `cernlib graflib`
    chmod u+x /tmp/$argv[1]
    breaksw

    case -e:
    /tmp/$argv[1]
    breaksw

    default:
    echo "Unknown option $argv[2]"
endif
```

Sulla base di questo esempio vi sarà possibile creare procedure di comandi che semplifichino le operazioni che eseguite frequentemente.

3.11 Archiviazione di files

È in generale buona norma archiviare e salvare il lavoro fatto su un apposito supporto per evitare che vada perso per erronee operazioni dell' utente o problemi di sistema. Nel corso non vi dovrete preoccupare di questo problema perchè l' archiviazione di files verrà eseguita automaticamente; tuttavia nel seguito descriveremo le operazioni di base necessarie.

3.11.1 Compressione di files

Per limitare l' occupazione dello spazio si disco può essere utile comprimere, con un processo riproducibile, i files in modo che occupino meno spazio. Per esemplificare il metodo di compressione consideriamo un file di testo: invece di scrivere tutti i caratteri si può scrivere il singolo carattere e quante volte questo è ripetuto (basti pensare a quanti spazi bianchi consecutivi sono contenuti in un file di testo per capire quanto si può guadagnare). Ovviamente gli attuali algoritmi di compressione utilizzano metodi più sofisticati di quello descritto.
Il comando per comprimere[2] è

`gzip` *fil*

che crea il file compresso *fil*.gz, che può essere decompresso con il comando:

`gunzip` *fil*.gz

Per archiviare un' intera struttura di directory in unico file si usa il comando:

`tar cvf` *tarfile.tar dir1 dir2 dir3*

che crea (`c`=create) il file archivio *tarfile*.
Per riottenere la struttura di directories dal file *tarfile.tar*:

`tar xvf` *tarfile.tar dir1 dir2 dir3*

(`x`=extract).
Il file di archivio può essere automaticamente compresso se si aggiunge l' opzione `z`.

[2] i comandi riportati sono modifiche elaborate dalla fondazione GNU ai comandi standard `zip` e `unzip`

3.11.2 Accesso a dischi rimuovibili

Uno dei supporti più semplici e più economici su cui salvare files sono i floppy disk.
Prima di utilizzare un floppy disk occore creare su di esso un "file-sistem" (formattazione) con il comando:
`mkfs -t` *os device blocks*
dove *os* è il tipo di "file-system" (*ext2* è il default in Linux, *msdos* è invece utile se volete rileggere il dischetto anche da Windows), *device* è l' indetificatore del device dell' unità floppy (in genere e nel nostro caso /dev/fd0) e *blocks* è il numero di blocchi (in unità di 1Kb) da allocare sul disco (per un floppy *blocks* = 1440).
Fatta questa operazione il dischetto è pronto per essere scritto ma non fa ancora parte del sistema su cui state lavorando; per includerlo nel sistema occorre montarlo con il comando:
`mount /mnt/floppy` (se il "file-system" è ext2),
`mount /mnt/winfloppy` (se il "file-system" è msdos).
A questo punto potete fare sul dischetto tutte le operazioni che usualmente fate sul disco rigido (quello che contiene la vostra HD).
Finite le operazioni dovete smontare il dischetto con il comando `umount` (analogo a `mount`).

3.12 Come ottenere ulteriori informazioni

Ulteriori informazioni sono reperibili, oltre che sui manuali cartacei, direttamente sul computer. La sintassi di ogni comando e funzione di libreria di Linux è documentata per mezzo di un help "on–line" cui potete accedere tramite il comando **man** (o **xman** se state utilizzando un X-terminal; la sintassi è

```
man <comando>
```

o semplicemente

```
xman &
```

Se non si sa il nome del comando che interessa, ma si conosce qualche parola chiave relativa all'argomento si può dare il comando

```
man -k parola_chiave
```

3.13 Il sistema grafico X11

Per utilizzare un elaboratore è necessario avere un terminale. Il terminale in senso stretto è oggetto "stupido", nel senso che si limita a mostrare sullo schermo i caratteri che riceve dal computer cui è collegato e a trasmettere al computer i caratteri digitati sulla tastiera.
Più spesso, però, quello che si indica con terminale è un piccolo computer capace di elaborazioni grafiche piuttosto complesse: in generale possiede un "sistema a finestre" dove ogni finestra può emulare un terminale tradizionale che può essere connesso ad uno o piu' elaboratori. Inoltre può ricevere comandi grafici da altri computers ed eseguirli mostrando il risultato su una finestra.
L' architettura "client-server" gestisce, nei sistemi LINUX, il funzionamento dei terminali.
Il "server" è un computer che invia dei comandi grafici, il "client" è un "oggetto" che riceve i comandi grafici e li esegue mostrando il risultato su terminale grafico. La comunicazione tra client e server avviene tramite una rete di trasmissione dati, nella quale ad ogni macchina è attribuito un "indirizzo" che ne consente l' identificazione (vedere Fig. 3.3).
Il protocollo di comunicazione si chiama X11.

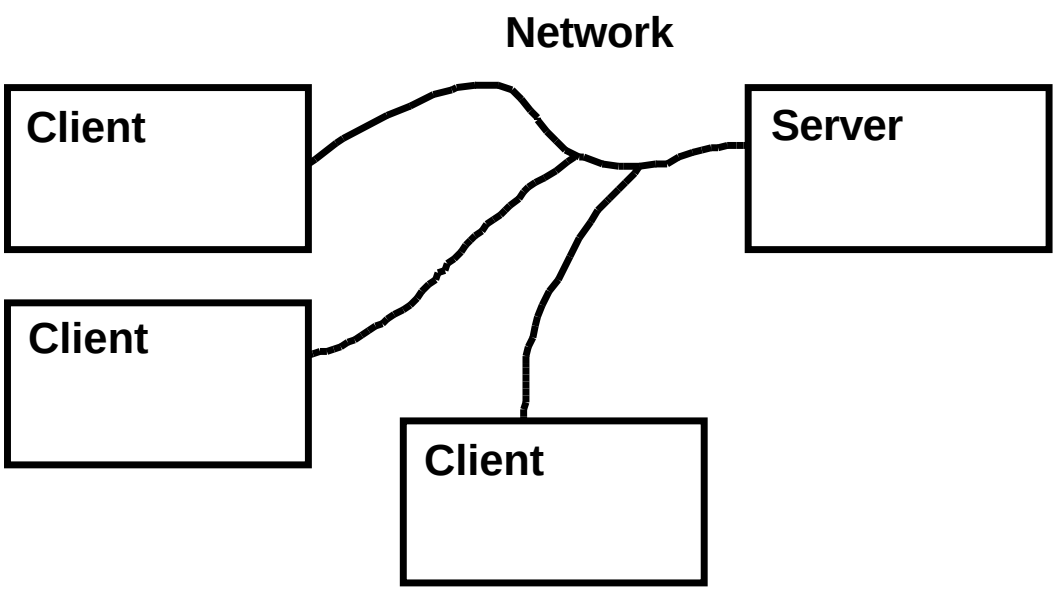

Figura 3.3: Esempio di sistema client-server

Per le esercitazioni utilizzerete dei PC con il sistema operativo LINUX.
Il loro indirizzo è **stationN.aula.fisica** con $N = 1..18$. In questo caso non si tratta di terminali ma di veri e propri computers che possono essere usati, oltre che per collegarsi ad un sistema remoto, anche per eseguire programmi localmente. In questo caso ciascuna macchina è un sistema "client-server".
Per utilizzare questi PC come semplici "client" nel collegamento remoto (as es. a server3) occorre seguire i seguenti passi:

- su una delle finestre inviare il comando

  ```
  telnet server3.fisica.unige.it
  ```

- Seguire la normale procedura di login (vedere Par 3.2.2).
- Informare **server3** dell' indirizzo dell' X-terminal che state usando, in modo che sia in grado di aprire le varie finestre grafiche su quel terminale. In generale questa operazione viene eseguita automaticamente durante il login; in alcuni casi però è necessario eseguirla manualmente. Se avete dubbi potete controllare quale X-terminal **server3** pensa voi stiate usando con il comando **echo $DISPLAY**, e fornire l' indirizzo con il comando **setenv DISPLAY xxxxN.fisica.unige.it:0** (Per maggiori dettagli vedere Par. 3.10)

Tutte le esercitazioni del corso utilizzeranno come i PC macchine locali, in questo caso la variabile DISPLAY viene automaticamente attribuita.
Altre macchine a vostra disposizione sono i PC dell'aula di Laboratorio 1 esp1N.fisica.unige.it $N = 01..10$ e i PC dell' aula terminali al quarto piano linuxdN.fisica.unige.it $N = 1..8$. Tutti questi PC hanno il sistema operativo Linux e condividono la stessa directory di login (vedi Par. 3.3).