

# Laboratorio di Sistemi Operativi

Alessandro Valenti

17 gennaio 2003



# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Obiettivi del corso . . . . .	4
<b>2</b>	<b>La shell di unix</b>	<b>7</b>
2.1	La sintassi della shell . . . . .	7
<b>3</b>	<b>Il filesystem</b>	<b>11</b>
<b>4</b>	<b>Uso delle utility</b>	<b>17</b>
<b>5</b>	<b>Strumenti dal sistema operativo</b>	<b>21</b>



# Capitolo 1

## Introduzione

Il corso di **Laboratorio di Sistemi Operativi** è un insegnamento a scelta per gli studenti di ingegneria elettronica ed informatica iscritti al secondo anno. La finalità del corso é quella di presentare una serie di esercizi ed esempi relativi all'uso di un sistema operativo per attività di programmazione.

Lo svolgimento dell'attività didattica è sincronizzato ed abbinato al corso di **Sistemi Operativi** in modo che, nel limite del possibile, gli studenti possano applicare in laboratorio concetti già visti e studiati teoricamente.

Per quanto si rimandi a detto corso di Sistemi Operativi ogni approfondimento sulla struttura di un sistema operativo, ritengo opportuno inserire una breve introduzione sull'argomento che potrà essere di aiuto agli studenti nella comprensione dei lavori successivi.

Un sistema operativo è, a tutti gli effetti, una porzione di software alla quale si delegano una serie di attività legate alla gestione delle risorse di un sistema programmato. Ogni personal computer, ad esempio, viene spesso equipaggiato con un sistema operativo direttamente dal costruttore o dall'installatore in quanto la maggior parte degli utenti si limiterà ad installare dei pacchetti software aggiuntivi già pronti per l'esecuzione delle più comuni attività informatiche.

Gli utenti più esperti, invece, possono provvedere alla sostituzione o all'installazione di altri sistemi operativi o, addirittura, alla realizzazione di programmi in grado di essere eseguiti senza sistema operativo.

L'inserimento di un sistema operativo è quindi una attività discrezionale dell'utente-programmatore che, tipicamente, dovrà prendere in considerazione pregi e difetti di entrambe le possibilità.

Un PC è, ad esempio, un sistema programmato discretamente complesso che richiede al programmatore che decidesse di utilizzarlo senza sistema operativo una grossa attività di sviluppo per utilizzarne tutte le potenzialità. L'entità del lavoro sarebbe invece molto minore per sistemi programmati *dedicati* o *embedded*, quali, ad esempio, sistemi di controllo industriale.

Il ruolo del sistema operativo non si limita solo alla gestione centralizzata delle periferiche, attività peraltro spesso dedicata ad un modulo software indipendente detto *bios*, ma coinvolge anche tutte le funzioni di messa in esecuzione e sincronizzazione di altri moduli software. Esistono sistemi operativi in grado di gestire, ad esempio, la coesistenza di più attività concorrenti (*task*) nella stessa CPU.

La progettazione e la realizzazione di alcuni programmi viene notevolmente semplificata se associata all'uso di sistemi operativi che consentono attività concorrenti detti, comunemente, sistemi operativi *multitasking*.

Il multitasking viene ottenuto alternando su base temporale l'esecuzione delle varie task; in altre parole ogni task ha a propria disposizione una porzione del tempo CPU. Una volta scaduta la porzione temporale a disposizione del task corrente, il contesto del task (memoria e stato dei registri CPU) viene congelato e sostituito con quello del nuovo task da eseguire che, a sua volta, era stato congelato precedentemente.

La determinazione dei *turni* di esecuzione è appannaggio dello **scheduler** che, di fatto, cerca di distribuire nel *miglior modo* il carico di lavoro.

Si noti, solo per completezza, che il *time-slicing* non è l'unico modo di ottenere il multitasking che, al contrario, può essere *preemptive* o *event-driven*.

Nell'utilizzo *storico* di unix il multitasking assume la particolare interpretazione di multi-processo in quanto ogni programma messo in esecuzione diviene un processo. Un processo è una entità astratta propria del sistema operativo che ha l'*illusione* di essere l'unico programma in esecuzione ed è associato all'utente che lo ha messo in esecuzione.

Nei sistemi programmati che presuppongono una interazione con l'utente, inoltre, il sistema operativo fornisce una interfaccia di controllo che consente di impartire comandi e di effettuare interrogazioni; questa interfaccia viene comunemente indicata con il termine *shell*.

La conoscenza di unix, pertanto, richiede una discreta padronanza dei concetti di utente e processo che lo studente otterrà attraverso una serie di esercitazioni relative alla programmazione, prima utilizzando la shell e, in seguito, il linguaggio C.

## 1.1 Obiettivi del corso

Le esercitazioni del corso di **Laboratorio di Sistemi Operativi** saranno svolte su macchine dotate di sistemi operativi appartenenti alla famiglia Unix quali i Pc del Laboratorio di Base (S.O. Linux) o le Sun del LICA (S.O. Solaris). Di conseguenza si farà riferimento alla shell ed alle funzioni di sistema proprie di Unix.

Gli studenti che volessero esercitarsi anche a casa, potranno installare una qualsiasi distribuzione Linux o BSD che, come è noto, non richiedono alcun acquisto di licenza d'uso.

Gli obiettivi prefissati per il corso di **Laboratorio di Sistemi Operativi** e che lo studente raggiungerà seguendo le lezioni in aula e mettendo in pratica le esercitazioni proposte per il laboratorio sono i seguenti (diapo: 1.2):

- capacità d'uso di macchine unix, della Bourne shell e dell'editor vi
- conoscenza della shell di unix e delle modalità di realizzazione di shell script
- padronanza dei sistemi di programmazione concorrente orientati ai processi e conoscenza dei principali metodi di sincronizzazione e comunicazione
- capacità di realizzare programmi in c utilizzando la libreria standard
- padronanza di unix come host di sviluppo

Università di Modena e Reggio Emilia	Laboratorio di Sistemi Operativi
<h3>Corso di Laboratorio di Sistemi Operativi</h3>	
corso facoltativo, CFU=3	
docente: Alessandro Valenti, valenti.alessandro@unimo.it	
Materiale didattico: dispense ed esercizi reperibili all'url: <a href="http://polaris.ing.unimo.it/didattica/labso">polaris.ing.unimo.it/didattica/labso</a>	
Modalità d'esame: esercitazione in abbinamento con quella di "Sistemi Operativi"	
Prova in itinere: una sulla programmazione shell e una sulla programmazione "c"	

Figura 1.1: front

Università di Modena e Reggio Emilia	Laboratorio di Sistemi Operativi
<h3>Obiettivi del Corso</h3>	
Capacità d'uso di macchine unix. della Bourne shell e dell'editor vi	
Conoscenza della shell di unix e delle modalità di realizzazione di shell script	
Padronanza dei sistemi di programmazione concorrenti orientati ai processi e conoscenza dei principali metodi di sincronizzazione e comunicazione	
Capacità di realizzare programmi in c utilizzando la libreria std	
Padronanza di unix come host di sviluppo	

Figura 1.2: obietti



## Capitolo 2

# La shell di unix

L'aspetto forse più interessante di unix è l'estrema modularità che contraddistingue tutti gli oggetti software che lo compongono. Il sistema operativo è di fatto costituito da un modulo software abbastanza ridotto che ne costituisce il kernel; ogni attività collaterale viene demandata ad opportuni programmi che si interfacciano al kernel mediante *chiamate di sistema* o *system calls*.

Anche una operazione apparentemente semplice come l'accesso di un utente al sistema coinvolge diversi componenti come indicato nella diapositiva in figura 2.2. A differenza di altri sistemi operativi, infatti, il kernel di unix non provvede direttamente all'interazione con l'utente ma si limita ad avviare un processo particolare di inizializzazione che rimarrà sempre attivo fino allo spegimento o *shutdown*; questo processo viene identificato con il nome **init** e, essendo il primo processo messo in esecuzione, sarà identificato con '0'.

Il processo **init**, a sua volta, provvede ad avviare altri processi secondo una sequenza definita in un opportuno file di configurazione; fra questi si trovano una o più occorrenze di processi *getty* ognuno dei quali prende controllo di un *terminale*.

Attraverso l'identificazione dell'utente, il processo *getty* ed il successivo processo *login* consentiranno l'accesso ai comandi tramite un interprete detto **shell**.

### 2.1 La sintassi della shell

La shell è un interprete di comandi che, come tale, viene caratterizzata da una c.d. grammatica. I comandi sono in pratica delle sequenze di caratteri che soddisfano la grammatica.

Il terminale collegato alla shell all'atto del login è un qualsiasi dispositivo in grado di produrre e ricevere caratteri, come la console (tastiera + video locali), una connessione seriale o una connessione remota via rete.

Un insieme di caratteri terminati da un ritorno a capo (in unix è il solo carattere LF) costituisce una *command line*. Il carattere particolare CTRL-D rappresenta la *fine file* e, di fatto, termina la sessione corrente.

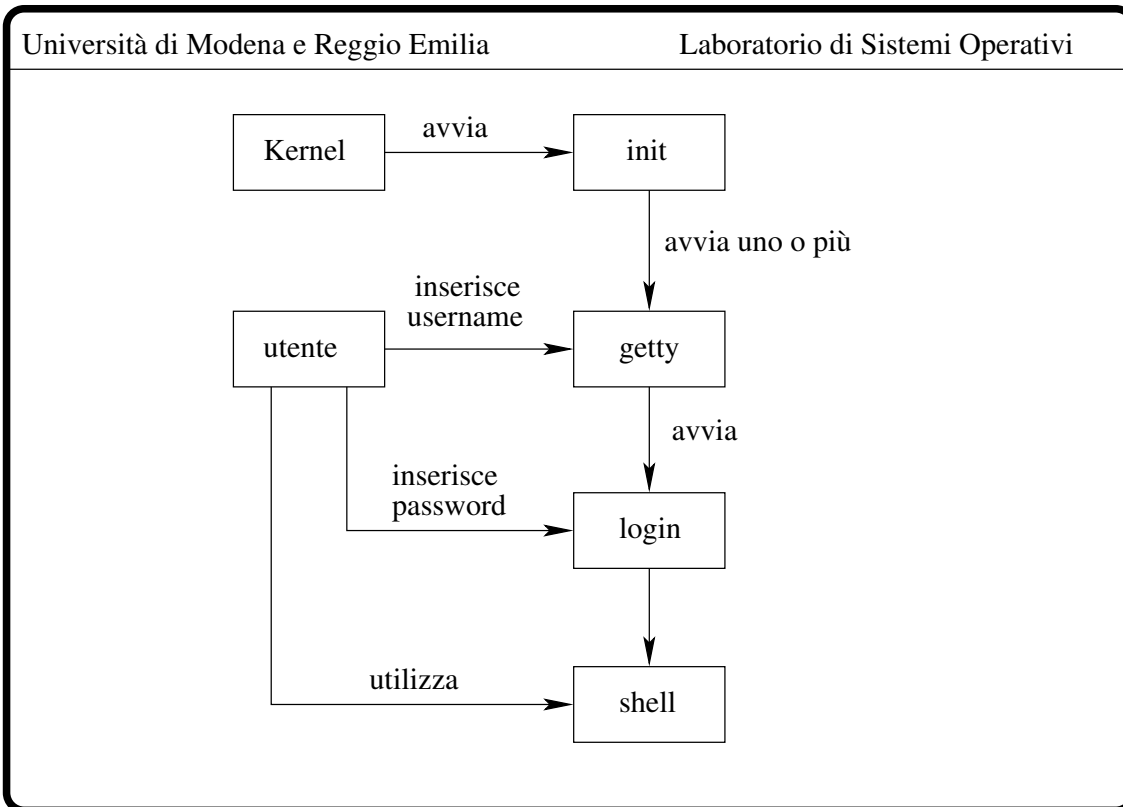


Figura 2.1: init

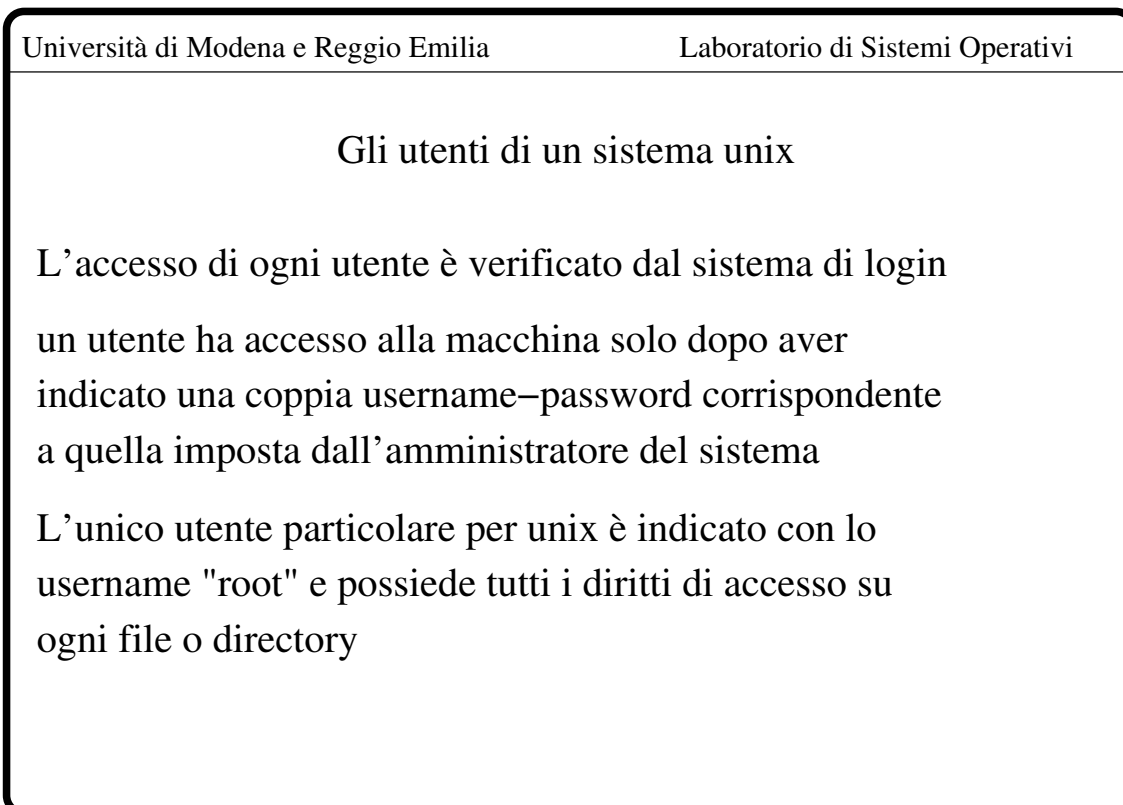


Figura 2.2: utenti

Università di Modena e Reggio Emilia	Laboratorio di Sistemi Operativi
--------------------------------------	----------------------------------

L'utente richiede l'esecuzione di un comando digitando una command line

Una command line è composta da uno o più caratteri terminati da un ritorno a capo (LF)

La shell interpreta ed esegue una o più command line fino alla ricezione del carattere CTRL-D o della command line "exit"

Gli elementi della linea di comando sono sequenze di caratteri separate dal carattere "spazio"

Il primo elemento corrisponde al nome di un comando o di un file eseguibile, gli elementi successivi sono gli argomenti

Figura 2.3: cmdline

Università di Modena e Reggio Emilia	Laboratorio di Sistemi Operativi
--------------------------------------	----------------------------------

Esempio di command line:

**ls**

elenca i file nella directory corrente

**ls -l**

elenca i file in formato "lungo", indicando più informazioni per ogni file. il "-l" è una opzione della command line passata al comando "ls"

**ls file1**

mostra, se presente nella directory corrente, il file con nome "file1". "file1" è un argomento della command line

Figura 2.4: ls

Ogni processo unix ha una propria directory corrente. Anche la shell è un processo e, di conseguenza, ha una directory corrente.

E' possibile cambiare la directory corrente della shell con il comando "cd nuovadir"

```
cd /
```

L'argomento "/" rappresenta la directory di destinazione che, in questo caso, è la directory "radice" o "root" del sistema.

L'insieme dei file e delle directory di un sistema ne costituisce il "file system". In unix tutti i file sono raggiungibili dalla radice "/" indicandone il "nome assoluto"

Figura 2.5: cd

E' possibile chiedere alla shell il nome assoluto della directory corrente con il comando:

```
pwd
```

Ogni file o directory può essere identificato con il proprio nome assoluto o con il percorso relativo alla directory corrente.

Se, ad esempio, la directory corrente è **/home/valealex**

sarà possibile raggiungere la dir /home/valealex/lso con uno qualsiasi dei seguenti comandi:

```
cd /home/valealex/lso
```

```
cd lso
```

Figura 2.6: pwd

## Capitolo 3

# Il filesystem

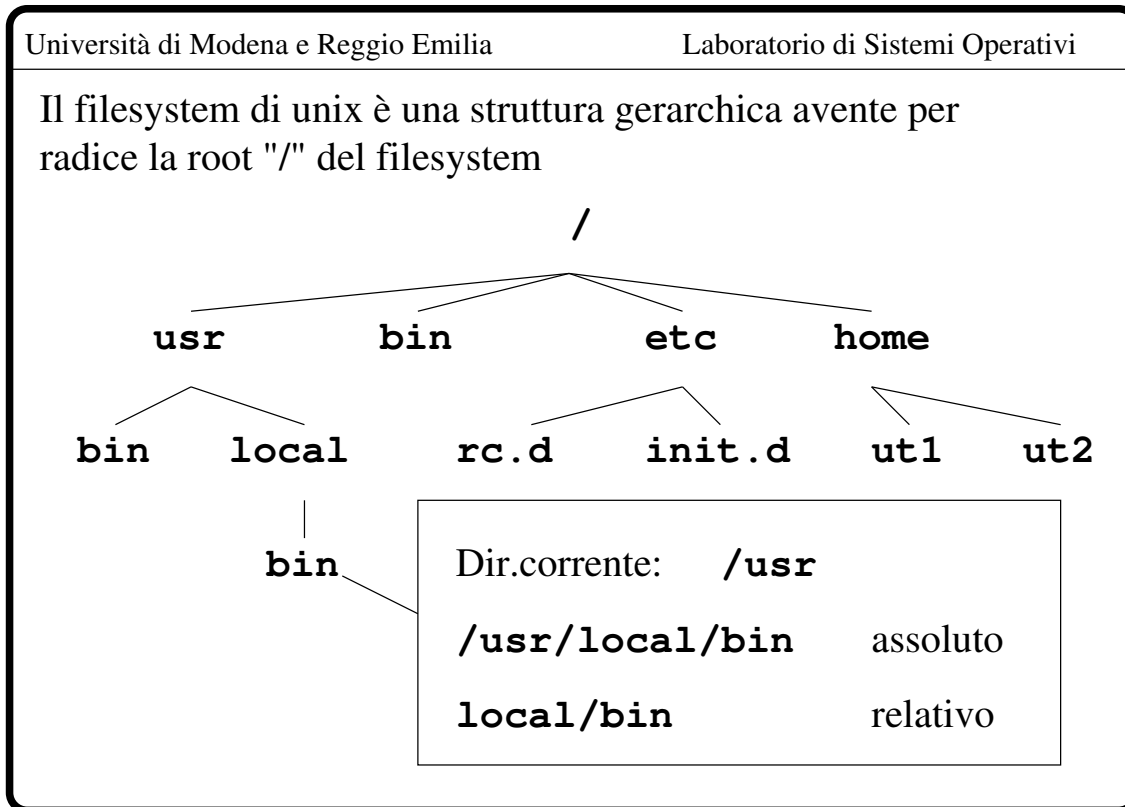


Figura 3.1: filesys

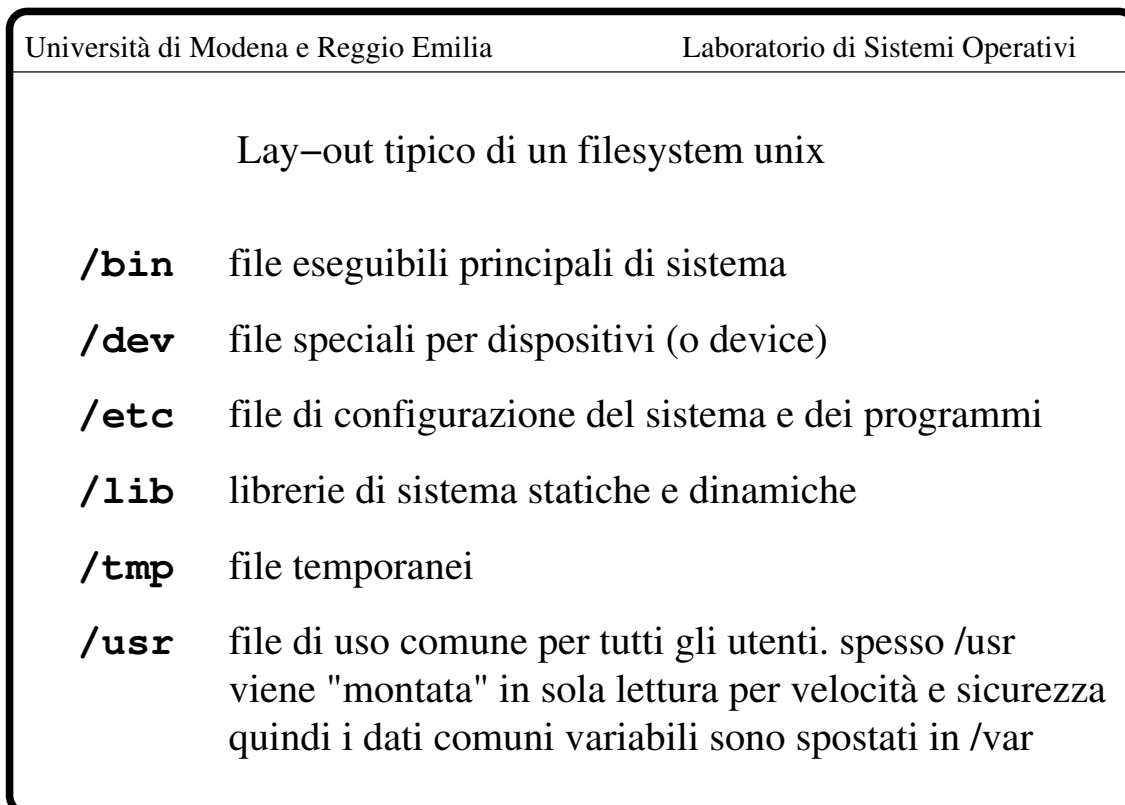


Figura 3.2: layout

Università di Modena e Reggio Emilia	Laboratorio di Sistemi Operativi
--------------------------------------	----------------------------------

la home directory

è una directory dedicata all'utente che normalmente ne possiede tutti i diritti e la proprietà

l'abbinamento fra nome utente e home directory è indicato nel file di configurazione `/etc/passwd`

la modifica, l'inserimento o la cancellazione di file dalla propria home directory non altera il funzionamento del sistema

l'utente può creare file o directory all'interno della propria home

ogni utente ha una propria home directory

Figura 3.3: home

Università di Modena e Reggio Emilia	Laboratorio di Sistemi Operativi
--------------------------------------	----------------------------------

proprietà dei file in unix

ogni file o directory in un filesystem unix è caratterizzato, oltre che dal nome, da alcune informazioni di accesso e protezione:

- identificativo del proprietario (utente)
- identificativo del gruppo (insieme di utenti)
- array di bit per i diritti di accesso

l'output del comando `"ls -l"` riporta queste informazioni per ogni file listato

Figura 3.4: fileprop

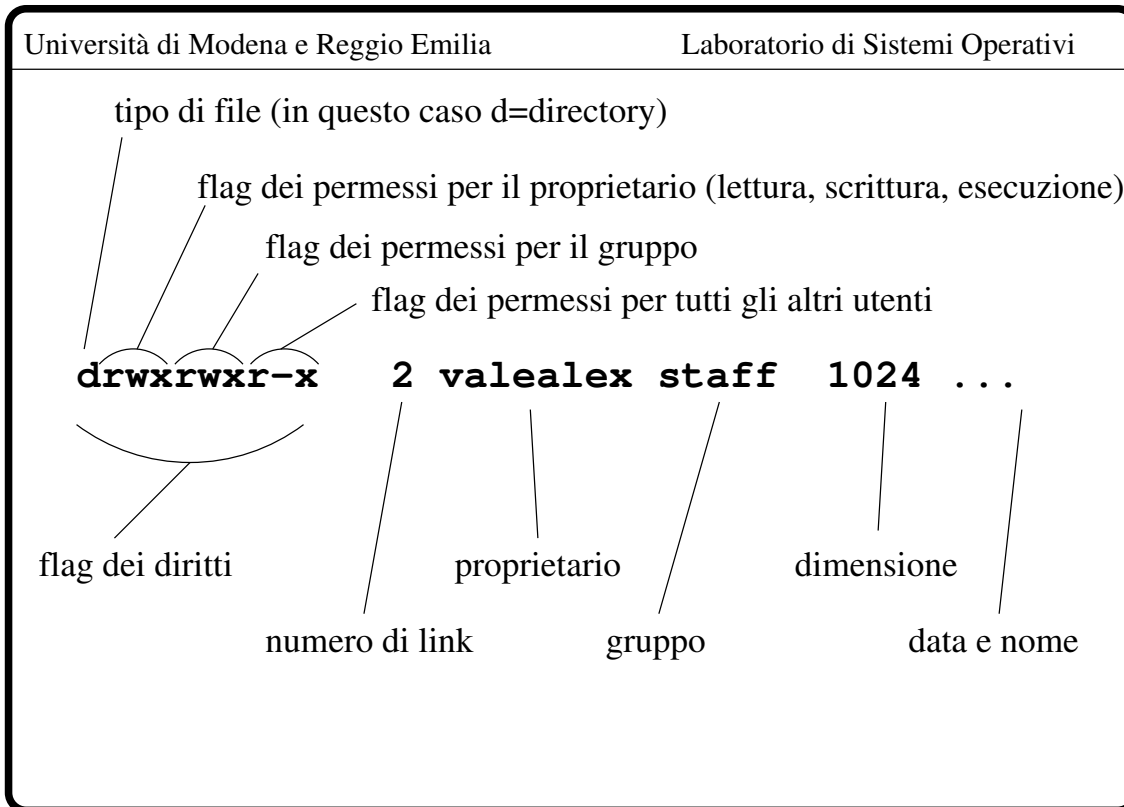


Figura 3.5: fileprop2

Università di Modena e Reggio Emilia Laboratorio di Sistemi Operativi

---

Oltre ai 9 bit relativi ai permessi di lettura, scrittura ed esecuzione per proprietario, gruppo e altri esistono 3 bit speciali per ogni file o directory in un filesystem unix:

	<b>s</b>		
--	----------	--	--

    suid: se settato su un file eseguibile, fornisce all'utente che lo mette in esecuzione i diritti del proprietario del file. Se settato su una directory propaga la proprietà della directory al proprietario della directory base

		<b>s</b>	
--	--	----------	--

    sgid: se settato su un file eseguibile, fornisce all'utente che lo mette in esecuzione i diritti del gruppo cui appartiene il file. Se settato su una directory propaga la il gruppo di appartenenza al gruppo che possiede la dir. base

			<b>t</b>
--	--	--	----------

    sticky: se settato su un file eseguibile impedisce lo swap su disco della memoria fisica occupata dal processo in esecuzione (in realtà questa funzionalità non è supportata dai moderni unix con una gestione già sicura della memoria tramite VM). Se settato su una directory (tipicamente /tmp) protegge ogni file creato in quella directory da scritture di utenti diversi dal proprietario anche se possiedono i diritti di scrittura sulla directory base.

Figura 3.6: suid

Quasi tutti i comandi disponibili in unix accettano delle opzioni che ne modificano il comportamento secondo regole specifiche ad ogni comando.

Anche il semplice 'ls' può essere utilizzato in modi diversi specificando opportune opzioni:

```
ls -l      mostra le informazioni dei file in formato "lungo"
ls -a      mostra tutti i file, anche quelli che iniziano con '.' normalmente esclusi
ls -A      come -a ma esclude i file particolari . e ..
ls -F      postpone il carattere '*' agli eseguibili e '/' alle directory
ls -d      lista il nome della directory senza listarne il contenuto (ls /tmp; ls -d /tmp)
ls -R      percorre ricorsivamente la gerarchia
ls -i      mostra anche il numero di inode per ogni file
ls -r      inverte l'ordine dell'elenco
ls -t      lista i file in ordine di modifica
ls -la     più opzioni possono essere combinate fra loro di seguito allo stesso '-' ...
ls -l -a   ... o con due opzioni distinte
```

Figura 3.7: ls2

I flag dei diritti di un file o di una directory esistente possono essere modificati da un utente che ne abbia il diritto di scrittura mediante il comando:

**chmod modifica nomefile**

l'argomento che indica la modifica da apportare ai diritti può essere espresso:

Simbolicamente, come variazione sui flag attuali:

```
chmod a+w file
chmod go-wx file
```

```
u=user      r=read      s=suid/sgid
g=group     w=write     t=sticky
o=others    x=execute
```

Con specificazione ottale della maschera desiderata:

```
chmod 0777 file
```

```
0100,0010,0001 read u,g,o
0200,0020,0002 write u,g,o
0400,0040,0004 execute
1000,2000,4000 suid,sgid
                sticky
```

Figura 3.8: chmod



## Capitolo 4

# Uso delle utility

Università di Modena e Reggio Emilia

Laboratorio di Sistemi Operativi

Il comando "cat" può essere utile per visualizzare un file o per creare un file con caratteri digitati da console.

**cat README**

visualizza a video il contenuto del file README

**cat**

visualizza a video ciò che si digita da tastiera

**cat >nuovofile**

memorizza i caratteri digitati da tastiera nel file "nuovofile" utilizzando il simbolo della shell ">" che rappresenta l'operazione di redirezione dell'output del comando.

Figura 4.1: catmore

Università di Modena e Reggio Emilia

Laboratorio di Sistemi Operativi

Utilizzando il comando 'cat' e la ridirezione è quindi possibile creare un file di testo:

**cat >file****ls -l****<CTRL>-z**

Ora possiamo rendere eseguibile il file da parte del proprietario:

**chmod u+x file**

e chiedere alla shell di eseguirne il contenuto

**./file**

Abbiamo in questo modo creato un semplice shell script che esegue il comando "ls -l" e termina.

Figura 4.2: catscript

La shell di unix, quindi, può invocare e mettere in esecuzione diversi comandi, ognuno dei quali può essere adattato alle specifiche esigenze mediante delle opzioni.

Ogni comando di unix mette a disposizione una documentazione, detta "manpage", che può essere visualizzata a console con il comando:

**man nomecomando**

L'uscita dal comando 'man' si ottiene con il carattere 'q'

Può essere conveniente stampare (vedere man man per i dettagli) le pagine dei comandi di uso più corrente.

Figura 4.3: man

Ogni sistema unix mette tipicamente a disposizione una serie di utility invocabili da shell:

<b>find</b>	cercare file nel filesystem
<b>grep</b>	cercare "pattern" o stringhe in un file
<b>sed/awk</b>	modificare file per cerca/sostituisci o su regole complesse
<b>tar/gzip/bzip2</b>	archiviare e comprimere file
<b>rcs/cvs</b>	controllo di versione
<b>make</b>	costruzione automatica di file e programmi
<b>cc/gcc</b>	compilazione sorgenti
<b>lex/yacc</b>	generazione codice
<b>vi/emacs</b>	editazione file
<b>man/info</b>	documentazione in linea

Figura 4.4: utility



## Capitolo 5

# Strumenti dal sistema operativo

La ridirezione è uno degli strumenti fondamentali per la programmazione in shell di unix

Ogni programma messo in esecuzione preleva caratteri da un "file" particolare detto standard input e produce risultati su un altro file particolare detto standard output.

Normalmente, un programma avviato da shell ha come stdin la tastiera e come stdout il video.

Questo comportamento può essere modificato con la ridirezione:

```
ls > file
ls >> file
cat < file
cat < file1 > file2
```

Figura 5.1: redir

Molti programmi utilizzano un altro file per i messaggi (output) di errore o, comunque, non inerenti al funzionamento principale del comando. Questo è identificato come stdout e la ridirezione di questo canale si ottiene con il simbolo:

```
comando 2> file
```

E' possibile ridirigere un canale su un altro, utilizzando il carattere '&' seguito dal numero identificativo del canale destinazione:

```
comando 2>&1
```

Le ridirezioni possono essere anche multiple:

```
comando >file 2>&1
comando 2>&1 >file
```

In lab, verificare cosa accade in questi due casi, usando come comando che genera flusso su stderr:

```
ls fileinesistente
```

Figura 5.2: redir2

Il sistema operativo, oltre a dare accesso al filesystem mediante syscall opportunamente invocate dalla shell o dai comandi precedenti, provvede a fornire gli strumenti per la gestione del multitasking tramite primitive sui processi.

Ogni processo unix è identificato univocamente sulla macchina da un numero intero detto "pid".

Il comando:

**ps**

elenca i processi attivi in varie modalità selezionabili da opportune opzioni.  
(in lab: vedere la manpage e verificare i risultati)

Nell'elenco dei processi è visualizzato anche il pid relativo ad ogni processo attivo.

(in lab: provare ps sh ps exit ps)

Figura 5.3: pid

una particolare struttura messa a disposizione dal sistema operativo unix che coinvolge sia la gestione dei file che quella dei processi è la pipe.

```
ps -ax > file
grep sh <file
rm file
```

questa sequenza di comandi salva l'output del programma "ps -ax" nel file e, in seguito, cerca la stringa "sh" all'interno del file stesso. Al termine il file è rimosso.

La concatenazione di programmi a mezzo file è controproducente per due motivi fondamentali:

- è scomodo creare il file di appoggio e rimuoverlo in seguito
- se la mole di dati è consistente, il secondo comando viene avviato dopo un certo ritardo mentre potrebbe iniziare ad elaborare "concorrentemente" al primo i dati già disponibili.

Figura 5.4: pipe

Università di Modena e Reggio Emilia

Laboratorio di Sistemi Operativi

utilizzando la concatenazione tramite pipe, i comandi possono essere concorrenti ed il sistema operativo si occupa di sincronizzare i processi.

```
ps -ax | grep sh
```

Anche nel caso in cui si volesse salvare su file una copia dei dati in transito fra due comandi, è conveniente usare la pipe per beneficiare della concorrenza fra i processi inserendo una "derivazione a T" ottenibile con il comando "tee":

```
ps -ax | tee file | grep sh
```

La pipe permette proprio la concatenazione di più comandi "modulari" collegando lo standard output del comando di sinistra con lo standard input del comando di destra.

Figura 5.5: pipe2