Sistemi Operativi[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

a.a. 2014/15

Sistemi Operativi

Bruschi Monga Re

Shell
Shell programming
Esercizi
I/O
Esercizi
Tabella riassuntiva
Shell e file system
File system

Lezione XVI: Unix power tools

Pipe

Sistemi Operativi

Bruschi Monga Re

Shell
Shell programming
Esercizi
I/O
Esercizi
Tabella riassuntiva
Shell e file system
File system

ls | sort

```
1  int main(void){
2          int fd[2], nbytes; pid_t childpid;
3          char string[] = "Hello, world!\n";
4          char readbuffer[80];
5
6          pipe(fd);
7          if(fork() == 0){
8                  /* Child process closes up input side of pipe */
9                  close(fd[0]);
10                 write(fd[1], string, (strlen(string)+1));
11                 exit(0);
12         } else {
13                 /* Parent process closes up output side of pipe */
14                 close(fd[1]);
15                 nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
16                 printf("Received string: %s", readbuffer);
17         }
18         return(0);}
```


Pipe (cont.)

Sistemi Operativi

Bruschi Monga Re

Shell
Shell programming
Esercizi
I/O
Esercizi
Tabella riassuntiva
Shell e file system
File system

```
1          if(fork() == 0)
2          {
3                  /* Close up standard input of the child */
4                  close(0);
5
6                  /* Duplicate the input side of pipe to stdin */
7                  dup(fd[0]);
8                  execlp("sort", "sort", NULL);
9          }
```

Un vero linguaggio di programmazione

La shell è un vero (Turing-completo) linguaggio di programmazione (interpretato)

- Variabili (create al primo assegnamento, uso con `$`, export in un'altra shell).
  - x="ciao"; y=2 ; /bin/echo "$x $y $x"
- Istruzioni condizionali (valore di ritorno 0 $\rightsquigarrow$ true)
  - **if** /bin/ls piripacchio; then /bin/echo ciao; **else** /bin/echo buonasera; fi
- Iterazioni su insiemi
  - **for** i in a b c d e; **do** /bin/echo $i; done
- Cicli
  - /usr/bin/touch piripacchio
  - **while** /bin/ls piripacchio; **do**
  - /usr/bin/sleep 2
  - /bin/echo ciao
  - done & (/usr/bin/sleep 10 ; /bin/rm piripacchio)

Esercizi

1. Per ciascuno dei file `dog`, `cat`, `fish` controllare se esistono nella directory `bin` (hint: usare `/bin/ls` e nel caso scrivere ``Trovato''`)
2. Consultare il manuale (programma `/usr/bin/man`) del programma `/bin/test` (per il manuale man test)
3. Riscrivere il primo esercizio facendo uso di `test`

Input e Output

In generale il paradigma UNIX permette alle applicazioni di fare I/O tramite:

Input

- Parametri al momento del lancio
- Variabili *d'ambiente*
- File (tutto ciò che può essere gestito con le syscall `open, read, write, close`)
  - Terminale (interfaccia testuale)
  - Device (per es. il mouse potrebbe essere `/dev/mouse`)
  - Rete (socket)

Output

- Valore di ritorno
- Variabili *d'ambiente*
- File (tutto ciò che può essere gestito con le syscall `open, read, write, close`)
  - Terminale (interfaccia testuale)
  - Device (per es. lo schermo in modalità grafica potrebbe essere `/dev/fb`)
  - Rete (socket)

Redirezioni

Ad ogni processo sono sempre associati tre file (già aperti)

- Standard input (Terminale, tastiera)
- Standard output (Terminale, video)
- Standard error (Terminale, video, usato per le segnalazione d'errore)

Possono essere *rediretti*

- /usr/bin/sort < lista Lo stdin è il file `lista`
- /bin/ls > lista Lo stdout è il file `lista`
- /bin/ls piripacchio 2> lista Lo stderr è il file `lista`
- (echo ciao & date ; ls piripacchio) 2> errori 1>output

Pipe

La pipe è un canale, analogo ad un file, bufferizzato in cui un processo scrive e un altro legge. Con la shell è possibile collegare due processi tramite una pipe anonima.
Lo stdout del primo diventa lo stdin del secondo

/bin/ls | sort

ls −lR / | sort | more

funzionalmente equivalente a

ls −lR >tmp1; sort <tmp1 >tmp2; more<tmp2; rm tmp∗

Molti programmi copiano lo stdin su stdout dopo averlo elaborato: sono detti filtri.

Command substitution

Con una pipe è possibile "collegare" lo stdout di un programma con lo stdin di un altro.
Per usare l'output di un programma sulla riga di comando di un altro programma, occorre usare la command substitution

/bin/ls −l $(/usr/bin/which sort)

Esercizi

1. Scrivere una *pipeline* di comandi che identifichi il le informazioni sul processo `dropbear` (`ps, grep`)
2. Scrivere una *pipeline* di comandi che identifichi il solo processo con il PPID piú alto (`ps, sort, tail`)
3. Ottenere il numero totale dei file contenuti nelle directory `/usr/bin` e `/var` (`ls, wc, expr`)
4. Si immagini di avere un file contenente il sorgente di un programma scritto in un linguaggio di programmazione in cui i commenti occupino intere righe che iniziano con il carattere #. Scrivere una serie di comandi per ottenere il programma senza commenti. (`grep`)
5. Ottenere la somma delle occupazioni dei file delle directory `/usr/bin` e `/var` (`du, cut`)

Tabella riassuntiva

Prog. (sez. man)	Descrizione
`ls (1)`	list directory contents
`echo (1)`	display a line of text
`touch (1)`	change file timestamps
`sleep (1)`	delay for a specified amount of time
`rm (1)`	remove files or directories
`cat (1)`	concatenate files and print on the standard output
`man (1)`	an interface to the on-line reference manuals
`test (1)`	check file types and compare values
`sort (1)`	sort lines of text files
`date (1)`	print or set the system date and time
`less (1)`	file perusal filter for crt viewing
`which (1)`	locate a command
`ps (1)`	report a snapshot of the current processes.
`tail (1)`	output the last part of files
`wc (1)`	print the number of newlines, words, and bytes in files
`test (1)`	check file types and compare values
`grep (1)`	print lines matching a pattern
`cut (1)`	remove sections from each line of files
`du (1)`	print disk usage

Link

- "A Brief Introduction to Unix (With Emphasis on the Unix Philosophy)", Corey Satten `http://staff.washington.edu/corey/unix-intro.pdf`
- `http://en.wikipedia.org/wiki/Unix_philosophy`
- "The UNIX Time-Sharing System", Ritchie; Thompson `http://www.cs.berkeley.edu/~brewer/cs262/unix.pdf`

Link

- `http://www.gnu.org/software/coreutils/manual/coreutils.html`

Ulteriori utility

Altre utility "standard" di cui è bene conoscere almeno l'esistenza

Prog. (sez. man)	**Descrizione**
`basename (1)`	strip directory and suffix from filenames
`dirname (1)`	strip non-directory suffix from file name
`uniq (1)`	report or omit repeated lines
`cut (1)`	remove sections from each line of files
`tr (1)`	translate or delete characters
`dd (1)`	convert and copy a file
`stat (1)`	display file or file system status

`cd` invece non è un programma, ma un comando interno della shell (che differenza fa?)

Shell e file system

- Ogni processo (compresa la shell stessa) ha associata una *directory di lavoro* (working directory), che può essere cambiata col comando (interno alla shell) `cd`
- I programmi fondamentali per operare sul file system

ls (1)	list directory contents
cp (1)	copy files and directories
rm (1)	remove files or directories
mv (1)	move (rename) files
mkdir (1)	make directories
rmdir (1)	remove empty directories
df (1)	report file system disk space usage
du (1)	estimate file space usage
pwd (1)	print name of current/working directory

Permessi

Ad ogni file vengono associati dei *permessi*, che definiscono le azioni permesse sui dati del file

- **Read:** leggere il contenuto del file o directory
- **Write:** scrivere (cambiare) il file o directory
- **eXecute** eseguire le istruzioni contenute nel file o accedere alla directory

R	W	X	
1	1	0	6
1	0	1	5
1	0	0	4
1	1	1	7

I permessi possono essere diversi per 3 categorie di utenti del sistema:

- **User:** il "proprietario" del file
- **Group:** gli appartenenti al gruppo proprietario
- **All:** tutti gli altri

Agire sui permessi

- Cambiare il proprietario
  - chown utente[:gruppo] file
- Cambiare il gruppo
  - chgrp gruppo file
- Cambiare i permessi
  - chmod 755 file
  - chmod +x file
  - chmod a=rw file
  - chmod g−x file
- (per creare un utente: adduser)

Il bit SUID

Il proprietario di un processo in esecuzione è normalmente *diverso* dal proprietario del file contente un programma (e diverso ad ogni esecuzione)

- effective UID bit: il processo assume come proprietario il proprietario del file del programma
- SUID `root`
- chmod 4555 file
- chmod u+s file