



Sistemi
Operativi

Bruschi
Monga Re

Shell

Esercizi
Shell
programming
Esercizi
I/O

Sistemi Operativi¹

Mattia Monga

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

a.a. 2015/16

¹ © 2008–16 M. Monga. Creative Commons Attribuzione — Condividi allo stesso modo 4.0 Internazionale. <http://creativecommons.org/licenses/by-sa/4.0/deed.it>. Immagini tratte da [2] e da Wikipedia.



Sistemi
Operativi

Bruschi
Monga Re

Shell

Esercizi

Shell
programming

Esercizi

I/O

Lezione XI: Processi, shell, file



Shell

La *shell* è l'*interprete dei comandi* che l'utente dà al sistema operativo. Ne esistono grafiche e testuali.

In ambito GNU/Linux la piú diffusa è una shell testuale `bash`, che fornisce i costrutti base di un linguaggio di programmazione (variabili, strutture di controllo) e primitive per la gestione dei processi e dei *file*.

Shell

Esercizi
Shell
programming
Esercizi
I/O



shell (pseudo codice)

Sistemi
Operativi

Bruschi
Monga Re

Shell

Esercizi
Shell
programming
Esercizi
I/O

```
1 while (1){ /* repeat forever */
2     type_prompt(); /* display prompt on the screen */
3     read_command(command, parameters); /* read input from terminal */
4     if (fork() > 0){ /* fork off child process */
5         /* Parent code. */
6         waitpid(1, &status, 0); /* wait for child to exit */
7     } else {
8         /* Child code. */
9         execve(command, parameters, 0); /* execute command */
10    }
11 }
```



Lanciare programmi con la shell

- Per iniziare l'esecuzione di un programma basta scrivere il nome del file
 - `/bin/ls`
- Il programma è trattato come una *funzione*, che prende dei parametri e ritorna un intero (`int main(int argc, char*argv[])`). Convenzione: 0 significa "non ci sono stati errori", > 0 errori (2 errore nei parametri), parametri - \rightsquigarrow opzioni
 - `/bin/ls /usr`
 - `/bin/ls piripacchio`
- Si può evitare che il padre aspetti la terminazione del figlio
 - `/bin/ls /usr &`
- Due programmi in sequenza
 - `/bin/ls /usr ; /bin/ls /usr`
- Due programmi in parallelo
 - `/bin/ls /usr & /bin/ls /usr`

Sistemi Operativi

Bruschi
Monga Re

Shell

Esercizi
Shell programming
Esercizi
I/O



- 1 Scrivere, compilare (`cc -o nome nome.c`) ed eseguire un programma che *forca* un nuovo processo.
- 2 Scrivere un programma che stampi sullo schermo `“Hello world! (numero)”` per 10 volte alla distanza di 1 secondo l'una dall'altra (`sleep(int)`). Terminare il programma con una chiamata `exit(0)`
- 3 Usare il programma precedente per sperimentare l'esecuzione in sequenza e in parallelo
- 4 Controllare il valore di ritorno con `/bin/echo $?`
- 5 Tradurre il programma in assembly con `cc -S -masm=intel nome.c`
- 6 Modificare l'assembly affinché il programmi esca con valore di ritorno 3 e controllare con `echo $?` dopo aver compilato con `cc -o nome nome.s`

POSIX Syscall (file mgt)



<code>fd = creat(name, mode)</code>	Obsolete way to create a new file
<code>fd = mknod(name, mode, addr)</code>	Create a regular, special, or directory i-node
<code>fd = open(file, how, ...)</code>	Open a file for reading, writing or both
<code>s = close(fd)</code>	Close an open file
<code>n = read(fd, buffer, nbytes)</code>	Read data from a file into a buffer
<code>n = write(fd, buffer, nbytes)</code>	Write data from a buffer into a file
<code>pos = lseek(fd, offset, whence)</code>	Move the file pointer
<code>s = stat(name, &buf)</code>	Get a file's status information
<code>s = fstat(fd, &buf)</code>	Get a file's status information
<code>fd = dup(fd)</code>	Allocate a new file descriptor for an open file
<code>s = pipe(&fd[0])</code>	Create a pipe
<code>s = ioctl(fd, request, argp)</code>	Perform special operations on a file
<code>s = access(name, amode)</code>	Check a file's accessibility
<code>s = rename(old, new)</code>	Give a file a new name
<code>s =fcntl(fd, cmd, ...)</code>	File locking and other operations

Sistemi
Operativi

Bruschi
Monga Re

Shell

Esercizi

Shell
programming

Esercizi

I/O

POSIX Syscall (file mgt cont.)



Sistemi
Operativi

Bruschi
Monga Re

<code>s = mkdir(name, mode)</code>	Create a new directory
<code>s = rmdir(name)</code>	Remove an empty directory
<code>s = link(name1, name2)</code>	Create a new entry, name2, pointing to name1
<code>s = unlink(name)</code>	Remove a directory entry
<code>s = mount(special, name, flag)</code>	Mount a file system
<code>s = umount(special)</code>	Unmount a file system
<code>s = sync()</code>	Flush all cached blocks to the disk
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chroot(dirname)</code>	Change the root directory

Shell

Esercizi

Shell
programming

Esercizi

I/O



```
1 int main(){
2     pid_t pid;
3     int f, off;
4     char string[] = "Hello, world!\n";
5
6     lsofd("padre (senza figli)");
7     printf("padre (senza figli) open *\n");
8     f = open("provaxxx.dat", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
9     if (f == -1){
10         perror("open");
11         exit(1);
12     }
13     lsofd("padre (senza figli)");
14     if (write(f, string, (strlen(string))) != (strlen(string)) ){
15         perror("write");
16         exit(1);
17     }
18
19     off = lseek(f, 0, SEEK_CUR);
20     printf("padre (senza figli) seek: %d\n", off);
21
22     printf("padre (senza figli) fork *\n");
23     if ( (pid = fork()) < 0){
24         perror("fork");
25         exit(1);
26     }
```



```
1  if (pid > 0){
2      lsofd("padre");
3      printf("padre write & close *\n");
4      off = lseek(f, 0, SEEK_CUR);
5      printf("padre seek prima: %d\n", off);
6      if (write(f, string, (strlen(string))) != (strlen(string)) ){
7          perror("write");
8          exit(1);
9      }
10     lsofd("padre");
11     off = lseek(f, 0, SEEK_CUR);
12     printf("padre seek dopo: %d\n", off);
13     close(f);
14     exit(0);
15 }
16 else {
17     lsofd("figlio");
18     printf("figlio write & close *\n");
19     off = lseek(f, 0, SEEK_CUR);
20     printf("figlio seek prima: %d\n", off);
21     if (write(f, string, (strlen(string))) != (strlen(string)) ){
22         perror("write");
23         exit(1);
24     }
25     lsofd("figlio");
26     off = lseek(f, 0, SEEK_CUR);
27     printf("figlio seek dopo: %d\n", off);
28     close(f);
29     exit(0);
30 }
31 }
```



Per fare esperimenti con i file descriptor può essere utile una funzione come la seguente

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3 #define _POSIX_SOURCE
4 #include <limits.h>
5
6 void lsofd(const char* name){
7     int i;
8     for (i=0; i<_POSIX_OPEN_MAX; i++){
9         struct stat buf;
10        if (fstat(i, &buf) == 0){
11            printf("%s fd:%d i-node: %d\n", name, i, (int)buf.st_ino);
12        }
13    }
14 }
```

Sistemi
Operativi

Bruschi
Monga Re

Shell

Esercizi

Shell
programming

Esercizi

I/O



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`
 - 2 `while /bin/ls piripacchio; do`
 - 3 `/usr/bin/sleep 2`
 - 4 `/bin/echo ciao`
 - 5 `done & (/usr/bin/sleep 10 ; /bin/rm piripacchio)`



- 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`

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)



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`



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);}
```

Sistemi
Operativi

Bruschi
Monga Re

Shell

Esercizi

Shell
programming

Esercizi

I/O



```
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  }
```



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**.



Sistemi
Operativi

Bruschi
Monga Re

Shell

Esercizi

Shell
programming

Esercizi

I/O