



Sistemi Operativi

Bruschi Monga

Astrazioni

Shell

Esercizi

file e pipe

# Sistemi Operativi<sup>1</sup>

Mattia Monga

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

a.a. 2013/14

<sup>1</sup> © 2008–14 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

Astrazioni

Shell

Esercizi

file e pipe

# Lezione VIII: Shell 2



Sistemi Operativi

Bruschi Monga

Astrazioni

Shell

Esercizi

file e pipe

# Astrazioni fornite dal s.o.

Le principali sono:

- System call
- Memoria virtuale
- Processo
- File
- Shell



Sistemi Operativi

Bruschi Monga

Astrazioni

Shell

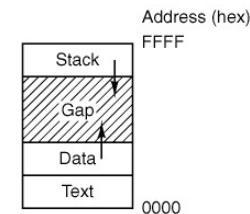
Esercizi

file e pipe

# Memoria virtuale

Il programmatore è libero di considerare un unico spazio di memoria, interamente dedicato al suo programma. Questo spazio può anche essere superiore alla memoria fisicamente disponibile.

Generalmente la memoria virtuale è divisa in *segmenti*: testo (codice), dati inizializzati, stack e heap.



## Processo



### Programma

Un programma è la codifica di un algoritmo in una forma eseguibile da una macchina specifica.

### Processo

Un processo è un programma in esecuzione.

### Thread

Un thread (*filo conduttore*) è una sequenza di istruzioni in esecuzione: più thread possono condividere lo spazio di memoria in cui le istruzioni lavorano. Il termine assume anche un'accezione tecnica nei sistemi operativi che distinguono le due astrazioni.

Ogni processo dà vita ad **almeno** un thread. Ogni CPU in un dato istante può eseguire **al più** un thread.

165

Sistemi Operativi  
Bruschi Monga  
Astrazioni  
Shell  
Esercizi  
file e pipe

## POSIX Syscall (process mgt)



<code>pid = fork()</code>	Create a child process identical to the parent
<code>pid = waitpid(pid, &amp;statloc, opts)</code>	Wait for a child to terminate
<code>s = wait(&amp;status)</code>	Old version of waitpid
<code>s = execve(name, argv, envp)</code>	Replace a process core image
<code>exit(status)</code>	Terminate process execution and return status
<code>size = brk(addr)</code>	Set the size of the data segment
<code>pid = getpid()</code>	Return the caller's process id
<code>pid = getpgid()</code>	Return the id of the caller's process group
<code>pid = setsid()</code>	Create a new session and return its process group id
<code>l = ptrace(req, pid, addr, data)</code>	Used for debugging

166

Sistemi Operativi  
Bruschi Monga  
Astrazioni  
Shell  
Esercizi  
file e pipe

## 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 più 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.

167

Sistemi Operativi  
Bruschi Monga  
Astrazioni  
Shell  
Esercizi  
file e pipe

## shell (pseudo codice)



```
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 }
```

168

Sistemi Operativi  
Bruschi Monga  
Astrazioni  
Shell  
Esercizi  
file e pipe

## 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 - ~> 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`

169

Sistemi Operativi

Bruschi Monga

Astrazioni

Shell

Esercizi  
file e pipe

## Esercizi



- 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 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`
- 7 Modificare l'assembly in modo che usi `scanf` per ottenere il numero di saluti.

170

Sistemi Operativi

Bruschi Monga

Astrazioni

Shell

Esercizi  
file e pipe

## 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, &amp;buf)</code>	Get a file's status information
<code>s = fstat(fd, &amp;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(&amp;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

171

Sistemi Operativi

Bruschi Monga

Astrazioni

Shell

Esercizi  
file e pipe

## POSIX Syscall (file mgt cont.)



<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

172

Sistemi Operativi

Bruschi Monga

Astrazioni

Shell

Esercizi  
file e pipe



```

1 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) pipe *\n");
8     f = open("provaxxx.dat", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
9     if (f == -1){
10         perror("pipe");
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     }

```

173

Sistemi  
OperativiBruschi  
Monga

Astrazioni

Shell

Esercizi

file e pipe



```

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

```

174

Sistemi  
OperativiBruschi  
Monga

Astrazioni

Shell

Esercizi

file e pipe

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(void){
7     int i;
8     for (i=0; i<_POSIX_OPEN_MAX; i++){
9         struct stat buf;
10        if (fstat(i, &buf) == 0){
11            printf("fd:%d i-node: %d\n", i, buf.st_ino);
12        }
13    }
14 }

```

175

Sistemi  
OperativiBruschi  
Monga

Astrazioni

Shell

Esercizi

file e pipe

Sistemi  
OperativiBruschi  
Monga

Astrazioni

Shell

Esercizi

file e pipe

633