



# Sistemi Operativi<sup>1</sup>

Mattia Monga

Dip. di Informatica e Comunicazione  
Università degli Studi di Milano, Italia  
[mattia.monga@unimi.it](mailto:mattia.monga@unimi.it)

a.a. 2008/09

<sup>1</sup> © 2009 M. Monga. Creative Commons Attribuzione-Condividi allo stesso modo 2.5 Italia License.  
<http://creativecommons.org/licenses/by-sa/2.5/it/>. Immagini tratte da [?] e da Wikipedia.



# Lezione V: Shell 1



# Cos'è un sistema operativo

## Sistema Operativo

Un s.o. è un programma che rende conveniente l'uso dello hardware

- fornendo astrazioni che semplificano l'uso delle periferiche e della memoria
- gestendo opportunamente le risorse fra tutte le attività in corso



# Astrazioni fornite dal s.o.

Le principali sono:

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

## System call



Sistemi Operativi

Bruschi Martignoni Monga

Le astrazioni del s.o.  
MINIX syscall

Shell  
Esercizi  
Shell programming  
Esercizi

Una chiamata di sistema (*syscall*) è la richiesta di un servizio al sistema operativo, che la porterà a termine in conformità alle sue *politiche*.

Per il programmatore è analoga ad una chiamata di procedura. Generalmente viene realizzata con un'*interruzione software* per garantire la protezione del s.o..

112

## Memoria virtuale



Sistemi Operativi

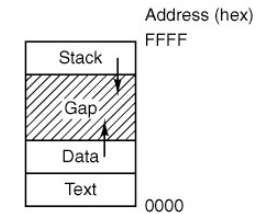
Bruschi Martignoni Monga

Le astrazioni del s.o.  
MINIX syscall

Shell  
Esercizi  
Shell programming  
Esercizi

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.

Minix fornisce una memoria virtuale divisa in *segmenti*: testo (codice), dati inizializzati, stack e heap.



113

## Processo



Sistemi Operativi

Bruschi Martignoni Monga

Le astrazioni del s.o.  
MINIX syscall

Shell  
Esercizi  
Shell programming  
Esercizi

### 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 il flusso di controllo sequenziale relativo ad un processo.

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

114

## File



Sistemi Operativi

Bruschi Martignoni Monga

Le astrazioni del s.o.  
MINIX syscall

Shell  
Esercizi  
Shell programming  
Esercizi

Un file è un insieme di byte conservato sulla memoria di massa. Hanno associato un nome e altri attributi.

In Minix i file sono organizzati gerarchicamente in *directory* (l'equivalente dei folder di MS Windows), che non sono che altri file contenenti un elenco.

115

La shell è l'*interprete dei comandi* che l'utente dà al sistema operativo. Ne esistono grafiche e testuali. In Minix, il default è una shell testuale ash, che fornisce i costrutti base di un linguaggio di programmazione (variabili, strutture di controllo) e primitive per la gestione dei processi e dei file.

```
pid = fork()
pid = waitpid(pid, &statloc, opts)
s = wait(&status)
s = execve(name, argv, envp)
exit(status)
size = brk(addr)
pid = getpid()
pid = getpgp()
pid = setsid()
l = ptrace(req, pid, addr, data)
```

Create a child process identical to the parent  
Wait for a child to terminate  
Old version of waitpid  
Replace a process core image  
Terminate process execution and return status  
Set the size of the data segment  
Return the caller's process id  
Return the id of the caller's process group  
Create a new session and return its process group id  
Used for debugging

|                                   |   |
|-----------------------------------|---|
| s = sigaction(sig, &act, &oldact) | Define action to take on signals                |
| s = sigreturn(&context)           | Return from a signal                            |
| s = sigprocmask(how, &set, &old)  | Examine or change the signal mask               |
| s = sigpending(set)               | Get the set of blocked signals                  |
| s = sigsuspend(sigmask)           | Replace the signal mask and suspend the process |
| s = kill(pid, sig)                | Send a signal to a process                      |
| residual = alarm(seconds)         | Set the alarm clock                             |
| s = pause()                       | Suspend the caller until the next signal        |

```
fd = creat(name, mode)
fd = mknod(name, mode, addr)
fd = open(file, how, ...)
s = close(fd)
n = read(fd, buffer, nbytes)
n = write(fd, buffer, nbytes)
pos = lseek(fd, offset, whence)
s = stat(name, &buf)
s = fstat(fd, &buf)
fd = dup(fd)
s = pipe(&fd[0])
s = ioctl(fd, request, argp)
s = access(name, amode)
s = rename(old, new)
s =fcntl(fd, cmd, ...)
```

Obsolete way to create a new file  
Create a regular, special, or directory i-node  
Open a file for reading, writing or both  
Close an open file  
Read data from a file into a buffer  
Write data from a buffer into a file  
Move the file pointer  
Get a file's status information  
Get a file's status information  
Allocate a new file descriptor for an open file  
Create a pipe  
Perform special operations on a file  
Check a file's accessibility  
Give a file a new name  
File locking and other operations

## MINIX Syscall (file mgt cont.)



DICo

Sistemi Operativi

Bruschi Martignoni Monga

Le astrazioni del s.o.

MINIX syscall

Shell

Esercizi  
Shell programming  
Esercizi

|   |  |
|---|--|
| <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                    |

120

## MINIX Syscall (protection)



DICo

Sistemi Operativi

Bruschi Martignoni Monga

Le astrazioni del s.o.

MINIX syscall

Shell

Esercizi  
Shell programming  
Esercizi

|  |                                 |
|--|---------------------------------|
| <code>s = chmod(name, mode)</code>         | Change a file's protection bits |
| <code>uid = getuid()</code>                | Get the caller's uid            |
| <code>gid = getgid()</code>                | Get the caller's gid            |
| <code>s = setuid(uid)</code>               | Set the caller's uid            |
| <code>s = setgid(gid)</code>               | Set the caller's gid            |
| <code>s = chown(name, owner, group)</code> | Change a file's owner and group |
| <code>oldmask = umask(complmode)</code>    | Change the mode mask            |

121

## MINIX Syscall (time)



DICo

Sistemi Operativi

Bruschi Martignoni Monga

Le astrazioni del s.o.

MINIX syscall

Shell

Esercizi  
Shell programming  
Esercizi

|   |   |
|---|---|
| <code>seconds = time(&amp;seconds)</code> | Get the elapsed time since Jan. 1, 1970   |
| <code>s = stime(tp)</code>                | Set the elapsed time since Jan. 1, 1970   |
| <code>s = utime(file, timep)</code>       | Set a file's "last access" time           |
| <code>s = times(buffer)</code>            | Get the user and system times used so far |

122

## Link



DICo

Sistemi Operativi

Bruschi Martignoni Monga

Le astrazioni del s.o.

MINIX syscall

Shell

Esercizi  
Shell programming  
Esercizi

- MINIX <http://www.minix3.org>
- Edsger W. Dijkstra, "My recollections of operating system design" <http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1303.PDF>

123



```

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 }
```



- 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



Bill Joy (co-fondatore della SUN), 1976, per BSD UNIX

- *Modal editor*
  - modo input
  - modo comandi
- I comandi di movimento e modifica sono sostanzialmente *ortogonali*
- small and fast
- fa parte dello standard POSIX



Salvare un file e uscire wq

- Modifica:
  - i, a insert before/after
  - o, O add a line
  - d, c, r delete, change, replace
  - y, p "to yank" and paste
  - u undo . redo
  - s/reg/rep/[g] search and replace
- Movimento:
  - h, j, k, l (o frecce)
  - O, beginning of line, \$, end of line
  - w, beginning of word, e, end of word
  - (num)G, goto line num, /, search
  - (, ), sentence



- 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)`) ed usarlo per sperimentare l’esecuzione in sequenza e in parallelo.
- 3 Controllare il valore di ritorno con `/bin/echo $?`

128



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 )`

129



- 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` (`man test`)
- 3 Riscrivere il primo esercizio facendo uso di `test`

130