



Sistemi Operativi¹

Mattia Monga

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

a.a. 2010/11



Lezione V: Shell 1



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



Le principali sono:

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

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

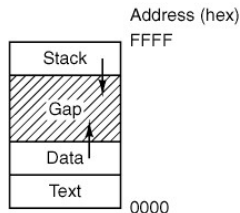


Una chiamata di sistema (*syscall*) è la richiesta di un servizio al sistema operativo, che la porterà a termine in conformità alle sue *politiche*.

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

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.





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.



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.



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.

MINIX Syscall (process mgt)



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

<code>pid = fork()</code>	Create a child process identical to the parent
<code>pid = waitpid(pid, &statloc, opts)</code>	Wait for a child to terminate
<code>s = wait(&status)</code>	Old version of <code>waitpid</code>
<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 = getpgrp()</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

MINIX Syscall (segnali)



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

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

MINIX Syscall (file mgt)



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

<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

MINIX Syscall (file mgt cont.)



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
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

MINIX Syscall (protection)



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

`s = chmod(name, mode)`

`uid = getuid()`

`gid = getgid()`

`s = setuid(uid)`

`s = setgid(gid)`

`s = chown(name, owner, group)`

`oldmask = umask(complmode)`

Change a file's protection bits

Get the caller's uid

Get the caller's gid

Set the caller's uid

Set the caller's gid

Change a file's owner and group

Change the mode mask

MINIX Syscall (time)



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

```
seconds = time(&seconds)
s = stime(tp)
s = utime(file, timep)
s = times(buffer)
```

Get the elapsed time since Jan. 1, 1970
Set the elapsed time since Jan. 1, 1970
Set a file's "last access" time
Get the user and system times used so far



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

shell (pseudo codice)



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

```
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



DICo

Sistemi
Operativi

Bruschi
Monga

Le astrazioni
del s.o.

MINIX syscall

Shell
Esercizi

- 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`



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)
 - 0, 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)`). 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 programma 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.