DICo

Sistemi Operativi[1]

Mattia Monga

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

a.a. 2011/12

Lezione V: Shell 1

Interruzioni

Un'interruzione (*interrupt request (IRQ)*) è un segnale (tipicamente generato da una periferica, ma non solo) che viene notificato alla CPU. La CPU, secondo le politiche programmate nel PIC, risponderà all'interruzione eseguendo il codice del gestore dell'interruzione (*interrupt handler*).
Dal punto di vista del programmatore la generazione di un'IRQ è analoga ad una chiamata di procedura, ma:

- Il codice è completamente disaccoppiato, potenzialmente in uno spazio di indirizzamento diverso (permette le protezioni)
- Non occorre conoscere l'indirizzo della procedura
- La tempistica dell'esecuzione è affidata alla CPU

BIOS (1/3)


```
1  ;Copyright (C) 2008 by Mattia Monga <mattia.monga@unimi.it>
2  bits 16 ; 16 bit real mode
3  org 0x7C00 ; origine indirizzo 0000:7C00
4
5  start:
6          cld ; clears direction flag (index regs incremented)
7          mov si, boot
8          call message
9  working:
10         mov si, work
11         call message
12
13         call waitenter
14         jmp working
```

BIOS (2/3)

```
1  message:
2          lodsb ; carica un byte da [DS:SI] in AL e inc SI
3          cmp al, 0
4          jz done
5          mov ah, 0x0E ; write char to screen in text mode
6          mov bx, 0 ; BH page number BL foreground color
7          int 0x10 ; write AL to screen (BIOS)
8          jmp message
9  done: ret
10
11 boot: db "Loading unuseful system...." , 10, 13, 0
12 work: db "I've done my unuseful stuff!" , 10, 13, 0
13 cont: db "Hit ENTER to continue...", 10, 13, 0
14 wow: db "Great! Hello world!" , 10, 13, 0
```

BIOS (3/3)

```
1  waitenter: mov si, cont
2                  call message
3                  mov ah, 0
4                  int 0x16 ; Wait for keypress (BIOS)
5                  cmp al, 'm'
6                  jz egg
7                  cmp al, 'b'
8                  jz basic
9                  cmp al, 13
10                 jnz waitenter
11                 ret
12 egg: mov si, wow
13                 call message
14                 jmp waitenter
15 basic: int 0x18  ; basic (BIOS)
16                 hlt
17
18           times 510-($-$$) db 0
19           dw 0xAA55
```

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

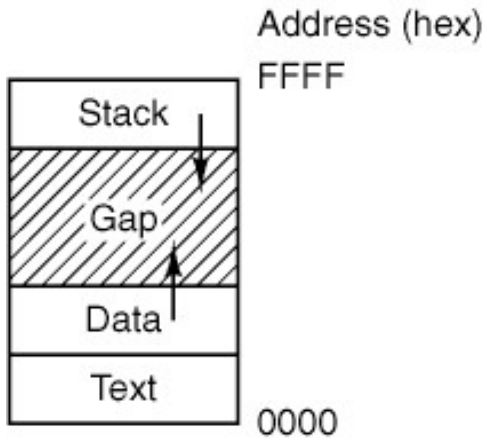
System call

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..

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.

Minix fornisce una memoria virtuale 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: piú 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 piú** un thread.

File

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.

Digressione: editor (di testo)

Editor

Un editor è un programma che permette di modificare arbitrariamente un *file*. Un editor di testo generalmente manipola file composto da caratteri stampabili.

- Emacs, vi
- nano, mined
- Notepad, Textpad, dots

Digressione: `vi`

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

vi in una slide

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

Shell

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)

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

MINIX Syscall (segnali)

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

MINIX Syscall (file mgt)

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

MINIX Syscall (file mgt cont.)

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

MINIX Syscall (protection)

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

MINIX Syscall (time)

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

Link

- 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)

```
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