Università di Verona

Corso di Laurea in Informatica/Informatica Multimediale
Laboratorio di Sistemi Operativi
09 Settembre 2008

Esercizio 1 (16 punti)
Scrivere uno script bash per rimuovere le IPC presenti nel sistema. Lo script deve ricevere come unico argomento la login dell'utente di cui si vogliono rimuovere le IPC. Una volta avviato, lo script deve far apparire un menu dal quale sia possibile scegliere il tipo di IPC da rimuovere come segue:
1 = Rimuovi i semafori
2 = Rimuovi le code di messaggi
3 = Rimuovi la memoria condivisa
4 = Esci dello script
Lo script deve visualizzare un messaggio di errore nel caso in cui l'utente digiti lettere o numeri non elencati nel menu, e deve uscire esclusivamente digitando il comando 4.

Esempio di output del comando ipcs

—— Shared Memory Segments ——

key	shmid	owner	perms	bytes	nattch	status
0x00000000	162070528	root	644	110592	3	dest

—— Semaphore Arrays——

key	shmid	owner	perms	nsems	status
0x0000004a	327680	root	666	3	

—— Message Queues——

key	msqid	owner	perms	used-bytes	messages
0x0000005a	0	root	666	0	0

Eventuali comandi da utilizzare:

```
cat [OPTION] [FILE]...
cut [OPTION]...  [FILE]...
echo [OPTION]...  [STRING]...
grep [options] PATTERN [FILE...]
ipcs [ -s | -m | -q ]
ipcrm [ shm | msg | sem ] id...
sort [OPTION]...  [FILE]...
```

Esercizio 2 (16 punti)
Utilizzando le system call di UNIX, scrivere un programma C per gestire il seguente problema. Un processo padre P crea due processi figli1 $F1$ e $F2$. Il processo $F2$, dopo un timeout di `TMOUT` secondi, invia un segnale a $F1$ e termina. $F1$, su ricezione del segnale, esegue il comando `ls -l`, stampa a video il risultato e segnala la fine del comando a P. Il padre P solo dopo la terminazione di $F1$ e $F2$ termina.

Eventuali System Call da utilizzare:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int pipe(int filedes[2]);
pid_t fork(void);
int kill(pid_t pid, int sig);
int execle(const char *path, const char *arg , ..., char * const envp[]);
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execv(const char *path, char *const argv[]);
```

```
int execvp(const char *file, char *const argv[]);
int dup2(int oldfd, int newfd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int close(int fd);
int dup(int oldfd);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
sighandler_t signal(int signum, sighandler_t handler);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```