Laboratorio di Sistemi Operativi
30 gennaio 2015
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) Il comando `cp` può prendere come primo argomento una lista di file? In caso di risposta affermativa ed in quel caso, il secondo argomento deve sottostare a qualche vincolo o può essere un qualunque file od una qualunque lista di file?

   Soluzione:

   Sì, il comando `cp` può prendere come primo argomento una lista di file. In questo caso il secondo argomento deve essere obbligatoriamente una directory.

2. (3 punti) Qual è l'effetto del seguente comando?

   ```
   tr abc ABC
   ```

   Scrivere un comando equivalente usando `sed`.

   Soluzione:

   Il comando `tr abc ABC` predispone la shell per accettare dell'input da parte dell'utente (infatti fa apparire il prompt secondario). Da quel momento fino alla pressione di `Ctrl+D` (fine file) tutte le linee inserite dall'utente verranno emesse sullo standard output con le lettere `a`, `b` e `c` convertite in maiuscolo.

   Un comando `sed` equivalente è

   ```
   sed y/abc/ABC/
   ```

3. (4 punti) L'output seguente mostra un frammento del contenuto del file `/etc/passwd`:

   ```
   root::0:0:root:/root:/bin/bash
   daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
   bin:x:2:2:bin:/bin:/usr/sbin/nologin
   mysql:x:124:134:MySQL Server,,,:/nonexistent:/bin/false
   sync:x:4:65534:sync:/bin:/bin/sync
   sys:x:3:3:sys:/dev:/usr/sbin/nologin
   ```

   Ogni linea contiene informazioni su un account del sistema in uso e tali informazioni sono formattate in campi separati dai due punti (`:`). Si scriva una pipeline che fornisca in output la lista dei soli nomi degli account (primo campo) ordinati numericamente in modo crescente in base allo user-id corrispondente (terzo campo). Ad esempio, per il frammento precedente l'output deve essere il seguente:

   ```
   root
   daemon
   bin
   sys
   sync
   mysql
   ```

   Soluzione:

   ```
   sort -t: -k3,4 -n /etc/passwd | cut -d: -f1
   ```

4. (5 punti) Si predisponga uno script della shell che legga da riga di comando una successione di numeri interi positivi terminata da `-1`. Lo script deve poi produrre sullo standard output un istogramma (utilizzando ad esempio il carattere `*`) tale che ogni linea sia lunga quanto il corrispondente numero della successione ricevuta in input (escluso il terminatore `-1` ovviamente). Si ignori la gestione degli eventuali errori. Esempio:

```
> ./istog.sh 3 5 1 6 -1
***
*****
*
******
```

Soluzione:

```
if test $# -eq 0
then
  echo "Usage: $0 n1 ... nk -1"
  exit 1
fi

n=$1

while ! test $n -eq -1
do
  i=0
  while test $i -lt $n
  do
    echo -n '*'
    i=$[$i+1]
  done
  echo
  shift
  n=$1
done

exit 0
```

5. (5 punti) Sia data la seguente struttura ricorsiva in C:

```
struct elemento {
  int valore;
  struct elemento *prossimo;
};
struct elemento *lista=NULL;
```

Si scriva il codice per aggiungere alla lista dinamica vuota puntata da `lista` gli elementi `2, 12, 6`.

Soluzione:

```
lista=(struct elemento*)malloc(sizeof(struct elemento));

if(lista==NULL) {
  perror("Allocazione di memoria fallita!\n");
```

```
  exit(1);
}

lista->valore=2;
lista->prossimo=(struct elemento*)malloc(sizeof(struct elemento));

if(lista==NULL) {
  perror("Allocazione di memoria fallita!\n");
  exit(1);
}

lista->prossimo->valore=12;
lista->prossimo->prossimo=(struct elemento*)malloc(sizeof(struct elemento));

if(lista==NULL) {
  perror("Allocazione di memoria fallita!\n");
  exit(1);
}

lista->prossimo->prossimo->valore=6;
lista->prossimo->prossimo->prossimo=NULL;
```

6. (6 punti) Si scriva un programma C che crei un processo figlio e si metta in attesa della terminazione di quest'ultimo. Il figlio tuttavia decide di terminare il processo padre inviandogli il segnale `SIGKILL`.

   Suggerimento: per ottenere il PID del padre si utilizzi la seguente system call:

```
SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       pid_t getppid(void);
DESCRIPTION: getppid() returns the process ID of the parent of the calling process.
```

Soluzione:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
  pid_t pid,ppid;

  switch(pid=fork()) {
  case -1:
    perror("Fork failed!\n");
    exit(1);
  case 0:
    printf("I am the son.\n");
    ppid=getppid();
```

```
    printf("Killing father (PID: %d)!\n",ppid);
    kill(ppid,SIGKILL);
    printf("Exiting (escaping :)\n");
    break;
  default:
    printf("I am the father.\n");
    waitpid(pid,NULL,0);
    printf("Son terminated!\n");
  }

  return 0;
}
```

7. (5 punti) Il programma seguente utilizza le funzioni sui semafori introdotte a lezione (i.e., `p()`, `v()` e `initsem()`) per creare `NUM_PROC` processi figli e consentire ad ognuno di essi di scrivere il proprio PID nel file `registro.txt` nella linea corrispondente (il primo processo figlio nella prima linea, il secondo nella seconda ecc.). Ogni linea è lunga `LENGTH` caratteri: l'ultimo carattere è il newline.

   Si completi il sorgente specificando i comandi mancanti da inserire al posto dei . . . nei 5 punti indicati, affinché un solo processo per volta possa accedere in modo esclusivo al file `registro.txt`.

```
void scrivi(key_t semkey,int fd,int linea) {
  int i,semid;
  char buffer[LENGTH];

  if((semid=initsem(semkey))<0) {
    perror("Errore nell'apertura del semaforo!\n");
    exit(1); }

  for(i=0;i<LENGTH-1;i++) buffer[i]=' '; // pulisce il buffer
  buffer[LENGTH-1]='\n'; // imposta il newline
  sprintf(buffer,"%d",getpid()); // scrive il PID nel buffer
  ... // <-- inizio sezione critica: completare (1)
  ... // sposta il puntatore di lettura/scrittura sulla linea giusta: completare (2)
  ... // scrive nel file: completare (3)
  ... // <-- fine sezione critica: completare (4)
}

int main() {
  key_t semkey=0x200;
  int fd,n,proc[NUM_PROC];

  fd=open("registro.txt",O_WRONLY | O_CREAT); // apre il file registro.txt in scrittura

  for(n=1;n<=NUM_PROC;n++) {
    switch(proc[n-1]=fork()) { // crea il figlio n-esimo
    case -1:
      perror("Fork fallita!\n");
      exit(1);
    case 0:
      scrivi(semkey,fd,n);
      exit(0);
    default:
      printf("Sono il padre %d: ho creato il figlio %d.\n",getpid(),proc[n-1]);
    }
  }
```

```
  for(n=1;n<=NUM_PROC;n++)
    ... // <-- il padre attende la terminazione del figlio n-esimo: completare (5)

  close(fd);
  return 0;
}
```

Soluzione:

Esempi di completamento dei punti indicati dall'esercizio:

1. `p(semid);`
2. `lseek(fd,(linea-1)*LENGTH,SEEK_SET);`
3. `write(fd,buffer,LENGTH);`
4. `v(semid);`
5. `waitpid(proc[n-1],NULL,0);`