Laboratorio di Sistemi Operativi
23 giugno 2015
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Si illustri la differenza fra link simbolico e link hard.

> **Soluzione:**
>
> In UNIX ogni entry di una directory è un link e rappresenta l'associazione fra un nome di file ed il corrispondente indice nell'array degli inode del dispositivo. Un link hard (creabile con il comando `ln`) ad un dato file è semplicemente un alias per lo stesso numero di inode del file. Al contrario, un link simbolico (creabile con il comando `ln -s`) ad un dato file è un file di testo (con un proprio inode diverso da quello del file a cui punta) trattato in modo speciale dal sistema operativo. Il file di testo contiene il path assoluto del file "puntato". Come conseguenza l'accesso tramite link hard ai file è molto più veloce rispetto all'accesso tramite link simbolici, ma questi ultimi consentono di far riferimento anche a file che risiedano su dispositivi e partizioni diversi.

2. Qual è l'effetto dei seguenti comandi?

    1. `f=~/.bash_profile`
    2. `echo 'basename $f'`
    3. `echo "basename $f"`
    4. `echo ` `basename $f` `

    Attenzione: nel punto 4 gli apici sono dei *backquote* (apici rovesciati).

> **Soluzione:**
>
> L'effetto dei comandi è il seguente:
>
> 1. `f=~/.bash_profile` assegna alla variabile `f` la stringa `/home/username/.bash_profile` dove `username` è il nome dell'account dell'utente che ha lanciato il comando.
> 2. `echo 'basename $f'` visualizza su standard output la stringa `basename $f` in quanto gli apici inibiscono l'interpretazione dei metacaratteri (compreso il `$`).
> 3. `echo "basename $f"` visualizza su standard output la stringa `basename /home/username/.bash_profile` dove `username` è il nome dell'account dell'utente che ha lanciato il comando (i doppi apici permettono l'interpretazione del metacarattere `$`).
> 4. `echo ` `basename $f` ` visualizza su standard output la stringa `.bash_profile` ovvero l'output del comando racchiuso tra gli apici rovesciati.

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

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
adm:x:3:4:adm:/var/adm:/bin/false
lp:x:4:7:lp:/var/spool/lpd:/bin/false
sync:x:5:0:sync:/sbin:/bin/sync
```

   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 successione di comandi che fornisca in output la lista dei soli nomi degli account (primo campo) e della relativa home directory (sesto campo) ordinati lessicograficamente in modo crescente in base al primo campo. Ad esempio, per il frammento precedente l'output deve essere il seguente:

```
adm     /var/adm
bin     /bin
daemon  /sbin
lp      /var/spool/lpd
root    /root
sync    /sbin
```

Soluzione:

Una possibile soluzione è la seguente:

```
sort -t: -k1,2 /etc/passwd | cut -d: -f1 > login.txt
sort -t: -k1,2 /etc/passwd | cut -d: -f6 > home.txt
paste login.txt home.txt
```

4. Si predisponga uno script della shell che prenda come argomento sulla linea di comando un intero positivo o nullo e ne calcoli il fattoriale, stampandolo su standard output. Si gestiscano gli eventuali errori relativamente a:

   1. passaggio di un numero di argomenti errato (ovvero, diverso da uno);
   2. passaggio di un intero negativo.

   Esempio:

```
> ./fact.sh 4
24
```

Soluzione:

Esempio di soluzione:

```
if test $# -ne 1
then
  echo "utilizzo: $0 n"
  exit 1
fi

if test $1 -lt 0
then
  echo "l'argomento deve essere un intero positivo o nullo"
  exit 2
fi

fact=1
n=$1

while test $n -gt 1
do
  fact=$[$fact * $n]
  n=$[$n - 1]
done

echo $fact

exit 0
```

5. Sia data la seguente struttura ricorsiva in C:

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

Si scriva il codice di una funzione avente il seguente prototipo:

```
void raddoppia(struct elemento *head);
```

che, scorrendo gli elementi della lista puntata da `head`, raddoppi il valore memorizzato nel membro `val` di ogni elemento.

Soluzione:
Esempio di soluzione:

```
void raddoppia(struct elemento *head) {
  while(head!=NULL) {
    head->val*=2;
    head=head->prossimo;
  }
}
```

6. Il programma seguente utilizza i thread ed i relativi meccanismi di accesso esclusivo (mutex) introdotti a lezione per creare `NUM_THR` thread figli e consentire ad ognuno di essi di scrivere il proprio THREAD ID nel file `registro.txt` nella linea corrispondente (il primo thread 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 thread per volta possa accedere in modo esclusivo al file `registro.txt`.

```
void *scrivi(void *n) { // il parametro n è l'indice corrispondente al thread
                        // ovvero, la sua posizione (linea) in registro.txt
  int i;
  char buffer[LENGTH];

  for(i=0;i<LENGTH-1;i++) buffer[i]=' '; // pulisce il buffer
  buffer[LENGTH-1]='\n'; // imposta il newline
  sprintf(buffer,"%lu",thread[((int)n)-1]); // scrive il THREAD ID 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)
};

pthread_mutex_t file_mutex=PTHREAD_MUTEX_INITIALIZER;

int main() {
  int n;
  fd=open("registro.txt",O_WRONLY | O_CREAT,0644); // apre il file registro.txt in scrittura
  for(n=1;n<=NUM_THR;n++) {
    if(pthread_create(&thread[n-1],NULL,scrivi,(void *)n)!=0) {
      perror("Errore nella creazione del thread.\n");
```

```
      exit(1);
    }
    printf("Sono il padre %d: ho creato il thread %lu.\n",getpid(),thread[n-1]);
  }
  for(n=1;n<=NUM_THR;n++)
    ... // <-- il padre attende la terminazione del figlio n-esimo: completare (5)
  close(fd);
  return 0;
}
```

Soluzione:

1. `pthread_mutex_lock(&file_mutex);`
2. `lseek(fd,(((int)n)-1)*LENGTH,SEEK_SET);`
3. `write(fd,buffer,LENGTH);`
4. `pthread_mutex_unlock(&file_mutex);`
5. `pthread_join(thread[n-1],NULL);`