Laboratorio di Sistemi Operativi
11 febbraio 2014
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Illustrare un modo per individuare qual è la propria shell di login.

   Risposta: Digitare al prompt della shell il comando seguente:

   ```
   echo $SHELL
   ```

2. Si supponga che la directory *corrente* `A` (contenente il file `f1`) si trovi nel filesystem del disco rigido, mentre la directory `B` si trovi nella radice del filesystem di una chiavetta USB (formattata con filesystem Unix) montata in `/mnt/usbmedia0`. Cosa succede se l'utente digita i comandi seguenti (si supponga che l'utente abbia tutti i privilegi necessari)?

   1. `ln f1 /mnt/usbmedia0/B/f1_link`
   2. `ln -s f1 /mnt/usbmedia0/B/f1_link`

   Risposta:

   1. Il tentativo di creazione del link hard fallisce in quanto link e file si troverebbero su due dispositivi fisici distinti (con array di inode distinti).
   2. Viene creato il link simbolico `f1_link` (all'interno della directory `B` nella radice della chiavetta USB) al file `f1` che risiede nel filesystem del disco rigido.

3. Qual è l'effetto dei seguenti comandi (nella sequenza fornita)?

   ```
   cd
   ls -al | grep '^d'
   ```

   Risposta: il primo comando (`cd`) sposta l'utente nella sua home directory. La pipeline successiva mostra in output su schermo soltanto le directory contenute nella home directory dell'utente. Infatti l'output di `ls -al` viene filtrato dal comando `grep` che "lascia passare" soltanto le linee che iniziano con il carattere `d`, ovvero, le linee corrispondenti alle directory.

4. Si predisponga uno script della shell che legga dallo standard input una serie di numeri interi, fermandosi quando incontra la stringa `stop` (su una linea da sola) e stampandone la somma su standard output. Si ignori la gestione degli eventuali errori.

   Risposta:

   ```
   read linea

   somma=0

   while test $linea != 'stop'
   do
     somma=$[$somma + $linea]
     read linea
   done

   echo $somma
   ```

5. Spiegare qual è l'effetto delle seguenti dichiarazioni di variabili in C:

   1. `int i;`
   2. `int *ip;`
   3. `int a1[10];`
   4. `int a2[]={0,1,2,3,4,5,6,7,8,9};`
   5. `int m[10][20];`
   6. `int *b[10];`

Risposta: L'effetto è il seguente:

1. viene dichiarata la variabile intera `i`;
2. viene dichiarato il puntatore `ip` ad interi;
3. viene dichiarato un vettore `a1` di `10` interi con indici da `0` a `9`.
4. viene dichiarato un vettore `a2` di 10 interi: il vettore viene anche inizializzato con i valori `0`, `1`, ..., `9`;
5. viene dichiarato un vettore `m` di `10`× `20` elementi interi (ovvero, una matrice di interi di`10` righe e `20` colonne);
6. viene dichiarato un vettore `b` di dieci puntatori ad interi;

6. Si scriva il codice C necessario per leggere dallo standard input delle quadruple di interi, memorizzando soltanto il primo ed il terzo di questi nelle variabili `x` e `y`, rispettivamente. La lettura avrà termine al momento in cui verrà rilevato l'end-of-file (`EOF`). Si ignori la gestione degli eventuali errori.

   Risposta:

   ```
   int x,y;

   while(scanf("%d %*d %d %*d",&x,&y)!=EOF);
   ```

7. Individuare l'errore nel seguente frammento di codice C:

   ```
   int *ip;
   printf("Il valore puntato da ip e': %d\n",*ip);
   ```

   Risposta: il puntatore `ip` viene utilizzato in un deriferimento senza essere stato inizializzato (errore logico).

8. Si assumano le seguenti direttive e dichiarazioni in C:

   ```
   #define MAX 81
   ...
   struct dati {
     int lunghezza;
     char testo[MAX];
   };
   struct dati *d;
   char s[]="Ciao, mondo!";
   ```

   Scrivere il codice per memorizzare (tramite il *puntatore* `d`) il valore di `s` e la sua lunghezza nei campi `testo` e `lunghezza`, rispettivamente.

   Risposta:

   ```
   d=(struct dati *)malloc(sizeof(struct dati));
   strcpy(d->testo,s);
   d->lunghezza=strlen(d->testo);
   ```

9. Il programma seguente dichiara una variabile `visit_log` (un array di caratteri) e lancia in esecuzione un numero`MAX` di thread. Ognuno di questi ultimi deve accedere a `visit_log`, stamparne il valore corrente ed aggiornarlo con la stringa ricevuta tramite la chiamata `pthread_create`, ovvero, `msg[i]` per il thread `i`-esimo.

   (a) 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 vettore di caratteri `visit_log`.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>

#define MAX 10
#define STRLEN 81

...                                                    # <- completare (punto 1)
char visit_log[STRLEN]="vuoto";

void *update_log(void *ptr);

main() {
  pthread_t thread[MAX];
  char msg[MAX][STRLEN];
  int i;

  for(i=0; i<MAX; i++) {
    sprintf(msg[i],"%s%d","Thread n.",i+1);
    printf("%s\n",msg[i]);

    if(pthread_create(&thread[i], NULL,
      (void *)&update_log, ...)!=0) {                   # <- completare (punto 2)
      fprintf(stderr,"Errore nella creazione del thread n. %d/%d.\n",i+1,MAX);
      exit(1);
    }

  }

  for(i=0; i<MAX; i++) {
    pthread_join(thread[i],NULL);
  }

  printf("Ultimo visitatore rilevato: %s\n",visit_log);
  exit(0);
}

void *update_log(void *ptr) {
  printf("%s - in attesa di accedere al log\n",(char *)ptr);
  ...                                                  # <- completare (punto 3)
  printf("%s - accesso al log; precedente visitatore rilevato: %s\n",(char *)ptr,visit_log);
  strcpy(visit_log, ...);                              # <- completare (punto 4)
  printf("%s - log aggiornato\n",(char *)ptr);
  ...                                                  # <- completare (punto 5)
  printf("%s - rilascio del log\n",(char *)ptr);
}
```

(b) Cosa potrebbe succedere nel caso in cui nel programma precedente venissero eliminate le seguenti linee del `main`?

```
for(i=0; i<MAX; i++) {
  pthread_join(thread[i],NULL);
}
```

Risposta:

(a) I punti vanno completati come segue:

1. `pthread_mutex_t log_mutex=PTHREAD_MUTEX_INITIALIZER;`
2. `(void *)msg[i]`
3. `pthread_mutex_lock(&log_mutex);`
4. `(char *)ptr`
5. `pthread_mutex_unlock(&log_mutex);`

(b) In assenza delle linee menzionate, il thread principale non attenderà la terminazione dei figli, provocando la fine prematura del processo (e di conseguenza di tutti i `MAX` thread creati).