Laboratorio di Sistemi Operativi
06 Luglio 2018
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) Si scriva una pipeline per estrarre dal file `/etc/passwd` il terzo campo di ogni riga, ovvero, il numero intero che rappresenta lo user-ID dell'utente relativo a quella riga (si ricordi che il separatore di campo del file `/etc/passwd` è il carattere *due punti*).

   La pipeline è la seguente:

   ```
   cat /etc/passwd | cut -d: -f3
   ```

2. (5 punti) Sfruttando la soluzione dell'esercizio precedente, si scriva uno script della shell che produca in output la somma di tutti gli user-ID contenuti nel file `/etc/passwd`.

   Suggerimento: si ricordi che `tail -n +k f` stampa le linee di `f`, a partire dalla `k`-esima, mentre `head -n +k f` stampa le prime `k` linee di `f`.

   ```
   num_linee=`wc -l < /etc/passwd`
   i=1
   somma=0

   while test $i -le $num_linee
   do
     linea=`cat /etc/passwd | head -n +$i | tail -1`
     valore=`echo $linea | cut -d: -f3 `
     somma=$[$somma+$valore]
     i=$[$i+1]
   done

   echo $somma
   ```

3. (16 punti) Si scriva un programma C che:
   (a) chieda all'utente di inserire il numero di righe e di colonne di una matrice;
   (b) allochi dinamicamente lo spazio in memoria per la matrice;
   (c) chieda all'utente di inserire gli elementi della matrice allocata;
   (d) stampi la matrice a video;
   (e) calcoli la trasposta della matrice;
   (f) stampi la matrice trasposta a video.

   Esempio di input/output:

   ```
   Inserisci il n. di righe e di colonne della matrice: 2
   3

   Inserisci gli elementi della matrice:
   Inserisci l'elemento a11: 2
   Inserisci l'elemento a12: 3
   Inserisci l'elemento a13: 4
   Inserisci l'elemento a21: 5
   Inserisci l'elemento a22: 6
   Inserisci l'elemento a23: 4

   Matrice inserita:
   2  3  4
   ```

```
5  6  4

Trasposta della matrice:
2  5
3  6
4  4
```

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

 int main()
 {
     int **a, **transpose;
     int r, c, i, j;
     printf("Enter rows and columns of matrix: ");
     scanf("%d %d", &r, &c);

     a=(int**)malloc(sizeof(int*)*r);
     transpose=(int**)malloc(sizeof(int*)*c);

     for(i=0; i<r; i++) {
        a[i]=(int *)malloc(sizeof(int)*c);
     }

     for(i=0; i<c; i++) {
        transpose[i]=(int *)malloc(sizeof(int)*r);
     }

     // Storing elements of the matrix
     printf("\nEnter elements of matrix:\n");
     for(i=0; i<r; ++i)
         for(j=0; j<c; ++j)
         {
             printf("Enter element a%d%d: ",i+1, j+1);
             scanf("%d", &a[i][j]);
         }

     // Displaying the matrix a[][] */
     printf("\nEntered Matrix: \n");
     for(i=0; i<r; ++i)
         for(j=0; j<c; ++j)
         {
             printf("%d  ", a[i][j]);
             if (j == c-1)
                 printf("\n\n");
         }

     // Finding the transpose of matrix a
     for(i=0; i<r; ++i)
         for(j=0; j<c; ++j)
         {
             transpose[j][i] = a[i][j];
         }

     // Displaying the transpose of matrix a
```

```
    printf("\nTranspose of Matrix:\n");
    for(i=0; i<c; ++i)
        for(j=0; j<r; ++j)
        {
            printf("%d  ",transpose[i][j]);
            if(j==r-1)
                printf("\n\n");
        }

    return 0;
}
```

4. (4 punti) Si dica cosa stampa il seguente programma C (giustificando la risposta):

```
#include <stdio.h>

int main() {
        int x;
        int *p;

        p=&x;
        x=1;
        printf("%d\n",(++x)+*p);
        *p=1;
        printf("%d\n",*p+(++x));
        return 0;
}
```

Il programma C stampa

```
4
3
```

Infatti, la prima espressione incrementa `x` a `2`, prima di valutarne il valore (grazie all'operatore di preincremento `++`), e poi ci somma nuovamente `2` in quanto `p` punta a `x`. La seconda espressione invece (dopo il reset di `x` a `1` per mezzo dell'assegnamento `*p=1`) valuta il contenuto della locazione di memoria puntata da `p`, ovvero, la locazione di `x`, ottenendo `1` a cui somma il valore di `x`, dopo aver incrementato quest'ultimo a `2` (grazie, nuovamente, all'operatore di preincremento).

5. (5 punti) Il programma seguente dichiara una variabile `visit_log` (un array di caratteri) e lancia in esecuzione un numero `MAX` di thread. Ognuno di essi 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.

   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);

int 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);
  return 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);
}
```

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);`