Laboratorio di Sistemi Operativi
06 Settembre 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*) e si ordini i valori in modo *crescente*.

   La pipeline è la seguente:

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

2. (5 punti) Sfruttando l'idea alla base della soluzione dell'esercizio precedente, si scriva uno script della shell che produca in output soltanto gli user-ID contenuti nel file `/etc/passwd` con un valore maggiore dell'argomento passato allo script sulla linea di comando.

   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`.

   Nel seguente esempio di soluzione si utilizza l'opzione `-r` di `sort` per ordinare in modo decrescente gli user-ID. In tal modo non sarà necessario scorrere tutte le linee del file (non appena il valore corrente risulterà minore od uguale al parametro passato su linea di comando, si potrà uscire dal ciclo, terminando l'esecuzione dello script).

```
num_linee=`wc -l < /etc/passwd`

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

if test $1 -lt -1
then
  echo "Utilizzo: $0 n # n>=-1"
  exit 2
fi

i=1
somma=0

while test $i -le $num_linee
do
  linea=`cat /etc/passwd | sort -n -r -t: -k3,3 | head -n +$i | tail
      -1`
  valore=`echo $linea | cut -d: -f3`
  if test $valore -gt $1
  then
    echo $valore
  else
    exit 0
  fi
  i=$[$i+1]
done

exit 0
```

Laboratorio di Sistemi Operativi
06 Settembre 2018
Compito

3. (13 punti) Si scriva un programma C che:
   (a) chieda all'utente di inserire il numero di righe e di colonne di una matrice quadrata;
   (b) allochi dinamicamente lo spazio in memoria per la matrice (usando `malloc()`);
   (c) chieda all'utente di inserire gli elementi della matrice allocata;
   (d) stampi la matrice a video;
   (e) calcoli e stampi a video il prodotto degli elementi della diagonale principale della matrice.

   Esempio di input/output:

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

Inserisci gli elementi della matrice:
Inserisci l'elemento a11: 2
Inserisci l'elemento a12: 3
Inserisci l'elemento a13: 5
Inserisci l'elemento a21: 5
Inserisci l'elemento a22: 6
Inserisci l'elemento a23: 3
Inserisci l'elemento a31: 9
Inserisci l'elemento a32: 1
Inserisci l'elemento a33: 7

Matrice inserita:
2  3  5
5  6  3
9  1  7

Prodotto degli elementi della diagonale principale della matrice:
84
```

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

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

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

     for(i=0; i<r; i++) {
        a[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<r; ++j)
         {
             printf("Enter element a%d%d: ",i+1, j+1);
             scanf("%d", &a[i][j]);
         }
```

```
25
26      // Displaying the matrix a[][] */
27      printf("\nEntered Matrix: \n");
28      for(i=0; i<r; ++i)
29          for(j=0; j<r; ++j)
30          {
31              printf("%d  ", a[i][j]);
32              if (j == r-1)
33                  printf("\n");
34          }
35
36      p=1;
37      for(i=0; i<r; ++i) {
38        p=p*a[i][i];
39      }
40
41      printf("Prodotto degli elementi della diagonale principale
           della matrice:\n%d\n",p);
42      return 0;
43  }
```

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

```
1  #include <stdio.h>
2
3  int main() {
4           int x;
5           int *p;
6
7           p=&x;
8           x=1;
9           printf("%d\n",(++x)*(*p));
10          *p=1;
11          printf("%d\n",(*p)*(++x));
12          return 0;
13 }
```

Il programma C stampa

```
4
2
```

Infatti, la prima espressione incrementa `x` a `2`, prima di valutarne il valore (grazie all'operatore di preincremento `++`), e poi lo moltiplica per `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` per cui moltiplica il valore di `x`, dopo aver incrementato quest'ultimo a `2` (grazie, nuovamente, all'operatore di preincremento).

5. (8 punti) Il codice seguente utilizza i thread ed i relativi meccanismi di accesso esclusivo (mutex) e di sincronizzazione (condition variable), per implementare una soluzione al problema classico dei produttori e consumatori con memoria limitata. Ci sono `NUM_P` thread produttori e `NUM_C` thread consumatori, che accedono in modo concorrente al vettore condiviso `buffer` con `LENGTH` posizioni per altrettanti interi positivi (che assumono valori da `1` a `MAX`). Per convenzione il valore `-1` indica

che la posizione del vettore è libera (vuota). Un thread produttore (se il buffer non è pieno) inserisce un nuovo elemento nella prima posizione libera. Un thread consumatore (se il buffer non è vuoto) preleva un elemento dalla prima posizione non vuota.

Supponendo di avere a disposizione

- la funzione `full()` che restituisce `1` se il buffer è pieno e `0` altrimenti,
- la funzione `empty()` che restituisce `1` se il buffer è vuoto e `0` altrimenti,

si completi il sorgente, specificando i comandi mancanti da inserire al posto dei ... negli 8 punti indicati, affinché il codice risultante rappresenti una soluzione corretta del problema dei produttori e consumatori con memoria limitata.

```
int buffer[LENGTH];       // buffer condiviso di lunghezza LENGTH
pthread_t threadP[NUM_P]; // vettore che contiene gli ID dei thread produttori
pthread_t threadC[NUM_C]; // vettore che contiene gli ID dei thread consumatori

// mutex per l'accesso esclusivo
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
// condition variable: buffer non vuoto
pthread_cond_t not_empty_buffer=PTHREAD_COND_INITIALIZER;
// condition variable: buffer non pieno
pthread_cond_t not_full_buffer=PTHREAD_COND_INITIALIZER;

void *producer(void *n) {
  int elemento, i;
  while(1) {
    printf("Thread produttore (ID %lu)\n",threadP[((int)n)-1]);
    ... // <-- inizio sezione critica: completare (1)
    ... // <-- controllo se posso inserire un nuovo elemento: completare (2)
    for(i=0; i<LENGTH; i++)
       if(buffer[i]==-1) {
         elemento=random()%MAX+1; // genero l'elemento
         buffer[i]=elemento;      // inserisco l'elemento
         break;
       }
    ... // <-- quale evento devo segnalare qui? completare (3)
    ... // <-- fine sezione critica: completare (4)
  }
};

void *consumer(void *n) {
  int elemento, i;
  while(1) {
    printf("Thread consumatore (ID %lu)\n",threadC[((int)n)-1]);
    ... // <-- inizio sezione critica: completare (5)
    ... // <-- controllo se posso prelevare un elemento (6)
    for(i=0; i<LENGTH; i++)
      if(buffer[i]!=-1) {
         elemento=buffer[i]; // prelevo l'elemento
         buffer[i]=-1;       // segnalo che la posizione è libera
         break;
      }
    ... // <-- quale evento devo segnalare qui? completare (7)
    ... // <-- fine sezione critica: completare (8)
  }
};
```

1. `pthread_mutex_lock(&mutex);`

2. `if(full()) pthread_cond_wait(&not_full_buffer,&mutex);`
3. `pthread_cond_signal(&not_empty_buffer);`
4. `pthread_mutex_unlock(&mutex);`
5. `pthread_mutex_lock(&mutex);`
6. `if(empty()) pthread_cond_wait(&not_empty_buffer,&mutex);`
7. `pthread_cond_signal(&not_full_buffer);`
8. `pthread_mutex_unlock(&mutex);`