Laboratorio di Sistemi Operativi
28 Giugno 2019
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (4 punti) Si scriva una pipeline che, partendo dalla directory corrente, stampi ricorsivamente i nomi delle directory leggibili ed attraversabili da tutte le categorie di utenti (ovvero, proprietario, gruppo e resto degli utenti).

```
find . -type d -ls | grep 'r.xr.xr.x'
```

2. (6 punti) si scriva uno script `explore.sh` della shell che accetti come argomento sulla linea di comando il percorso di una directory e la esplori ricorsivamente, stampando alla fine i valori di due contatori: uno relativo al numero di directory incontrate ed uno relativo al numero di file regolari incontrati.

   Esempio:

```
> ./explore.sh /home/pippo
number of dirs: 10
number of files: 23
> _
```

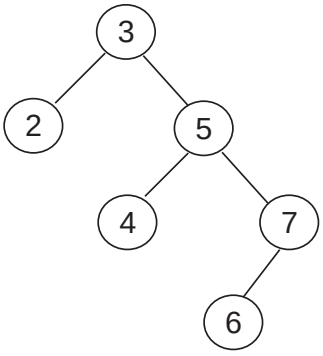
```
1  if test $# -ne 1
2  then
3          echo "Wrong number of arguments!"
4          exit 1
5  fi
6
7  if ! test -d $1
8  then
9          echo "$1 must be a directory!"
10         exit 2
11 fi
12
13 find $1 -type d > /tmp/dirs.txt
14 find $1 -type f > /tmp/files.txt
15
16 dirs=`cat /tmp/dirs.txt | wc -l`
17 files=`cat /tmp/files.txt | wc -l`
18
19 echo "number of directories: $dirs"
20 echo "number of files: $files"
21
22 rm /tmp/dirs.txt
23 rm /tmp/files.txt
24
25 exit 0
```

3. (8 punti) Data la seguente struttura dati ricorsiva `Node`:

```
struct node_t {
  int data;
  struct node_t *left;
  struct node_t *right;
};
```

```
typedef struct node_t Node;
```

si scriva un programma C che allochi spazio in memoria ed inizializzi il seguente albero binario (con radice nel nodo 3):



Si scriva infine una procedura ricorsiva che riceva come parametro il puntatore alla radice di un albero binario (come, ad esempio, quello precedente) e ne liberi la memoria occupata dai vari nodi.

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

struct node_t {
  int data;
  struct node_t *left;
  struct node_t *right;
};

typedef struct node_t Node;

void deltree(Node *r) {
        if(r!=NULL) {
                if(r->left!=NULL) deltree(r->left);
                if(r->right!=NULL) deltree(r->right);
                free(r);
        }
}

int main() {
  Node *root=NULL;
  root=(Node*)malloc(sizeof(Node));
  root->data=3;
  root->left=(Node*)malloc(sizeof(Node));
  root->right=(Node*)malloc(sizeof(Node));
  root->left->data=2;
  root->left->left=NULL;
  root->left->right=NULL;
  root->right=(Node*)malloc(sizeof(Node));
  root->right->data=5;
  root->right->left=(Node*)malloc(sizeof(Node));
  root->right->right=(Node*)malloc(sizeof(Node));
  root->right->left->data=4;
```

```
  root->right->left->left=NULL;
  root->right->left->right=NULL;
  root->right->right->data=7;
  root->right->right->left=(Node*)malloc(sizeof(Node));
  root->right->right->right=NULL;
  root->right->right->left->data=6;
  root->right->right->left->left=NULL;
  root->right->right->left->right=NULL;
  deltree(root);
  return 0;
}
```

4. (6 punti) Si dica qual è l'output generato dal seguente programma C:

```
#include <stdio.h>

int main() {
  int i,j;
  for(i=0; i<3; i++) {
    for(j=0; j<i; j++)
      printf(" ");
    for(j=0; j<(3-i)*2-1; j++)
      printf("|");
    printf("\n");
  }

  return 0;
}
```

```
|||||
 |||
  |
```

5. (8 punti) Il programma seguente utilizza i thread, i mutex e le 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 (con valori da `1` a `MAX`). Il valore `-1` indica che la posizione del vettore è libera (vuota). Un produttore (se il buffer non è pieno) inserisce un nuovo elemento nella prima posizione libera. Un 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 negli 8 punti indicati, affinché il codice risultante rappresenti una soluzione corretta del problema.

```
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);`