Laboratorio di Sistemi Operativi
8 Luglio 2021
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `check_file_tree.sh` che prenda come argomento sulla linea di comando un percorso, controlli che corrisponda ad una directory e attraversi ricorsivamente il file system a partire da essa, stampando il percorso dei soli file (quindi non delle sottodirectory) incontrati, la relativa dimensione in byte e, alla fine, il totale dei byte occupati da essi.

   Esempio:

   ```
   ./check_file_tree.sh .
   ./check_file_tree.sh, 374 byte
   ./draw.c, 197 byte
   ./test, 16864 byte
   ./dice.c, 1031 byte
   ./test.c, 738 byte
   ./draw, 16696 byte
   ./dice, 17272 byte
   Numero totale di byte: 53172
   ```

   Esempio di soluzione:

   ```
   1  if ! test $# -eq 1
   2  then
   3      echo "Utilizzo $0 pathname"
   4      exit 1
   5  fi
   6
   7  num_bytes=0
   8
   9  if test -e $1 -a -d $1
   10 then
   11     lista=‘find $1 -print 2>/dev/null‘
   12     for i in $lista
   13     do
   14       if test -f $i -a ! -l $i
   15       then
   16         dim=$(cat $i | wc -c)
   17         echo $i, $dim byte
   18         num_bytes=$[$num_bytes+$dim]
   19       fi
   20     done
   21 fi
   22
   23 echo "Numero totale di byte: $num_bytes"
   24
   25 exit 0
   ```

2. (6 punti) Si scriva una pipeline che stampi a video l'elenco (senza ripetizioni e ordinato lessicograficamente *al contrario*) dei nomi di login degli utenti di sistema.

   Suggerimento: si ricorra al file `/etc/passwd` dove ogni linea corrisponde ad un account del sistema ed è una successione di campi separati dai due punti (`:`). Il campo relativo al nome di login è il primo.

Esempio di soluzione:

```
cat /etc/passwd | cut -d ':' -f1 | sort -r | uniq
```

3. (6 punti) Si considerino le seguenti dichiarazioni in C:

```
struct lista {
  int n;
  struct lista *next;
};
struct binary_tree {
  int n;
  struct binary_tree *left;
  struct binary_tree *right;
};
struct lista *l;
struct binary_tree *t;
struct lista l2={5, NULL};
struct binary_tree t2={1, NULL, NULL};
int a[10]={1,1,1,1,1,1,1,1,1,1};
```

Si dica, per ognuna delle seguenti sequenze di comandi, se è corretta oppure no (motivando la risposta e correggendo gli eventuali errori):

Sequenza 1:

```
l=NULL;
t=NULL;
```

Sequenza 2:

```
t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
if(t!=NULL) {
  t->n=a[5];
  t->left=t->right=NULL;
}
```

Sequenza 3:

```
a[1]=l2->n;
```

Sequenza 4:

```
l=(struct lista *)malloc(sizeof(struct lista));
if(l!=NULL) {
  (*l).n=l2.n+1;
  l->next=l2;
}
```

Sequenza 5:

```
t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
if(t!=NULL) {
  t->n=a[5];
  t->left=&t2;
  t->right=&l2;
}
```

Sequenza 6:

```
t2.n=l2.n+1;
```

Risposte:

1. sequenza corretta: viene assegnata la costante `NULL` ad entrambi i puntatori `l` e `t` che così denotano, rispettivamente, la lista vuota e l'albero binario vuoto (senza nodi);
2. sequenza corretta: viene allocata memoria per un nodo di un albero binario, assegnando al puntatore `t` il suo indirizzo; in seguito viene assegnato l'intero contenuto nella sesta posizione dell'array `a` al campo `n` del nodo appena allocato (i campi `left` e `right` si vedono assegnare la costante `NULL`, ad indicare che il nodo è privo di figli);
3. sequenza scorretta: l'operatore `->` si applica solo ai puntatori a strutture (correzione: a[1]=l2.n;);
4. sequenza scorretta: a `l->next` bisogna assegnare un puntatore di tipo `struct lista *`. Correzione:

```
l=(struct lista *)malloc(sizeof(struct lista));
if(l!=NULL) {
  (*l).n=l2.n+1;
  l->next=&l2;
}
```

5. sequenza scorretta: il tipo di `&l2` è `struct lista *`, mentre dovrebbe essere `struct binary_tree *`. Correzione:

```
t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
if(t!=NULL) {
  t->n=a[5];
  t->left=&t2;
  t->right=NULL; // oppure: t->right=&t2;
}
```

6. sequenza corretta: all'intero `t2.n` viene assegnato il successore dell'intero `l2.n`.

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

```
#include <stdio.h>

int main() {
  int i,j;
  for(i=2; i>=0; i--) {
    for(j=0; j<i; j++)
      printf(" ");
    for(j=0; j<(5-i)*2-1; j++)
      printf("*");
    printf("\n");
  }

  return 0;
}
```

```
  *****
 *******
*********
```

Laboratorio di Sistemi Operativi
8 Luglio 2021
Compito

5. (10 punti) Si scriva un programma C `dice.c` che simuli il lancio di un dado. Il programma prende da linea di comando il numero `n` (maggiore o uguale a `1`) di thread che deve lanciare in esecuzione. Ogni thread inizia a generare un numero pseudo-casuale compreso tra 1 e 6 ogni 5 secondi. Il primo thread che genera il punteggio massimo provoca la terminazione di tutto il processo, stampando a video il messaggio `E' stato raggiunto il punteggio massimo!`.

   Suggerimento: si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
#include <stdlib.h>
#include <time.h>
...
srandom(time(NULL)); // per inizializzare il seme con la data/ora attuale, assicurandosi di generare sequenze diverse ad ogni esecuzione
...
long int r;
r=random(); // genera un numero pseudo-casuale con valore compreso tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di soluzione:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <pthread.h>

int stop=0;
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

void *dice(void *ptr) {
  while(!stop) {
    long int r=random();
    int score=r%6+1;

    if(score==6) {
      pthread_mutex_lock(&mutex);
      printf("Punteggio massimo raggiunto.\n");
      stop=1;
      pthread_mutex_unlock(&mutex);
    }

    sleep(5);
  }
}

int main(int argc, char** argv) {
  if(argc!=2) {
    fprintf(stderr,"Uso: %s n\n",argv[0]);
    return 1;
  }
  int num_thread=atoi(argv[1]);
  pthread_t *thread_id=NULL;

  if(num_thread>=1) {
    thread_id=(pthread_t*)malloc(sizeof(pthread_t)*num_thread);
```

```
36
37      if(thread_id!=NULL) {
38        for(int i=0; i<num_thread; i++) {
39          if(pthread_create(&thread_id[i],NULL,dice,NULL)!=0) {
40            fprintf(stderr,"Errore nella creazione del thread n. %d
                  .\n",i+1);
41            exit(1);
42          }
43        }
44      } else {
45          perror("Memoria insufficiente.\n");
46          exit(1);
47      }
48
49      for(int i=0; i<num_thread; i++)
50        pthread_join(thread_id[i],NULL);
51      free(thread_id);
52    }
53
54    return 0;
55  }
```