Laboratorio di Sistemi Operativi
9 Settembre 2021
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `del_files.sh` che prenda come argomento sulla linea di comando una stringa ed un percorso, controlli che quest'ultimo corrisponda ad una directory e attraversi ricorsivamente il file system a partire da essa, cancellando tutti i file incontrati che abbiano come estensione la stringa fornita come primo argomento. Durante la cancellazione deve stampare a video il percorso dei file cancellati e, alla fine, deve stampare il numero totale di file rimossi.

   Esempio:

   ```
   ./del_files.sh bak .
   ./a.bak
   ./b/c.bak
   Numero di file cancellati: 2
   ```

   Esempio di soluzione:

   ```
   1  if ! test $# -eq 2
   2  then
   3          echo "Utilizzo $0 extension pathname"
   4      exit 1
   5  fi
   6
   7  num_deleted_file=0
   8  files=‘find $2 -name "*.$1"‘
   9
   10 for f in $files
   11 do
   12     if test -f $f
   13     then
   14         echo $f
   15         rm -f $f
   16         num_deleted_file=$[$num_deleted_file+1]
   17     fi
   18 done
   19
   20 echo "Numero di file cancellati: $num_deleted_file"
   21
   22 exit 0
   ```

2. (6 punti) Si supponga che sia stata impostata una variabile di ambiente di nome `utente`. Si scriva un unico comando o pipeline, utilizzando i vari metacaratteri di composizione di comandi della shell, che stampi a video la stringa `ok` (e niente altro) se esiste un utente del sistema con nome di login uguale al valore della variabile `utente`. In caso contrario, il comando o pipeline non deve stampare nulla.

   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:

   ```
   1 cat /etc/passwd | cut -d: -f1 | grep "$utente" /etc/passwd 2>&1 >/
       dev/null && echo ok
   ```

Laboratorio di Sistemi Operativi
9 Settembre 2021
Compito

3. (8 punti) Scrivere il codice di un programma C che prenda come argomento sulla linea di comando il percorso di un file di testo e stampi a video il suo contenuto, applicando il seguente filtro: devono essere rimossi dal testo tutti i caratteri eccetto le lettere dell'alfabeto (maiuscole e minuscole). Si gestiscano inoltre gli eventuali errori (numero di argomenti errato, file non leggibile, ecc.).

```
#include <stdio.h>

#define SIZE 150

void filter(char *line) {
    int i,j;

    for(i = 0; line[i] != '\0'; ++i)
    {
        while (!( (line[i] >= 'a' && line[i] <= 'z') ||
                  (line[i] >= 'A' && line[i] <= 'Z') ||
                  (line[i] == '\0')
                ))
        {
            for(j = i; line[j] != '\0'; ++j)
            {
                line[j] = line[j+1];
            }
            line[j] = '\0';
        }
    }

    puts(line);
}

int main(int argc, char **argv) {
    char line[SIZE];
    int i, j;

    if(argc!=2) {
        fprintf(stderr, "Usage: %s file-path\n", argv[0]);
        return 1;
    }

    FILE *f=fopen(argv[1],"r");

    if(f==NULL) {
        fprintf(stderr, "file %s not found!\n", argv[1]);
        return 2;
    }

    while(fgets(line,SIZE,f)!=NULL) {
        filter(line);
    }

    fclose(f);

    return 0;
}
```

4. (4 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};
```

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=0;
t=0;
```

Sequenza 2:

```
t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
if(t!=NULL) {
  t->n=l2.n;
  t->left=t->right=0;
}
```

Sequenza 3:

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

Sequenza 4:

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

Risposte:

1. sequenza corretta: viene assegnata la costante `0` 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 nel campo `n` del nodo `l2` al campo `n` del nodo appena allocato (i campi `left` e `right` si vedono assegnare la costante `0`, ad indicare che il nodo è privo di figli);
3. 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;
}
```

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

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. Prima di terminare, il processo salva nel file `log.txt` il messaggio `Punteggio massimo raggiunto dal thread con ID: x.`, dove `x` è l'ID del thread che ha generato 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 esecuzione:

```
$ ./dice 8
2 (139833826105088)
4 (139833809319680)
5 (139833817712384)
2 (139833724888832)
6 (139833716496128)
$ cat log.txt
Punteggio massimo raggiunto dal thread con ID: 139833716496128.
```

Esempio di soluzione:

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

int stop=0;
pthread_t winner;
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

void *dice(void *ptr) {
  while(!stop) {
    long int r=random();
    int score=r%6+1;
    printf("%d (%lu)\n",score,*((pthread_t *)ptr));
    if(score==6) {
      pthread_mutex_lock(&mutex);
      stop=1;
      winner=*((pthread_t *)ptr);
      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]);
  winner=0;
  pthread_t *thread_id=NULL;

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

    if(thread_id!=NULL) {
      for(int i=0; i<num_thread; i++) {
        if(pthread_create(&thread_id[i],NULL,dice,&thread_id[i])
           !=0) {
          fprintf(stderr,"Errore nella creazione del thread n. %d
             .\n",i+1);
          exit(1);
        }
      }
    } else {
        perror("Memoria insufficiente.\n");
        exit(1);
    }

    for(int i=0; i<num_thread; i++)
      pthread_join(thread_id[i],NULL);

    FILE* log_file=fopen("log.txt","w");
    char buffer[80];
    sprintf(buffer,"Punteggio massimo raggiunto dal thread con ID:
        %lu.\n",winner);
    fwrite(buffer,sizeof(char),strlen(buffer),log_file);
    fclose(log_file);
    free(thread_id);
  }

  return 0;
}
```