Laboratorio di Sistemi Operativi
14 luglio 2015
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (a) Qual è il significato e l'importanza dell'*exit status* dei comandi e degli script della shell?
   (b) Si indichi un comando per visualizzare su standard output l'esito (exit status) dell'ultimo comando eseguito.

> **Soluzione:**
>
> (a) L'*exit status* di un comando o script della shell è un intero che rappresenta l'esito della sua esecuzione. In particolare, per convenzione, si stabilisce che un exit status pari a `0` indica una terminazione normale, mentre un exit status diverso da zero indica una terminazione con errore. Ciò è importante in quanto l'analisi di tale valore consente di eseguire delle scelte nel flusso di esecuzione dei comandi successivi (attraverso l'utilizzo dei costrutti di scelta condizionale e di iterazione negli script, ad esempio).
>
> (b) Un esempio di comando per visualizzare su standard output l'esito (exit status) dell'ultimo comando eseguito è il seguente:
>
> ```
> echo $?
> ```

2. Qual è l'effetto dei seguenti comandi?

   1. `lista=‘ls ~‘`
   2. `echo ’$lista’`
   3. `echo "$lista"`

   Attenzione: nel punto 1 gli apici sono dei *backquote* (apici rovesciati).

> **Soluzione:**
>
> 1. `lista=‘ls ~‘` esegue il comando `ls ~` e ne salva l'output (ovvero l'elenco dei file contenuti nella home directory dell'utente) come valore della variabile `lista`.
> 2. `echo ’$lista’` produce la visualizzazione su standard output della stringa `$lista`, dato che gli apici singoli inibiscono il metacarattere `$`.
> 3. `echo "$lista"` produce la visualizzazione su standard output del valore della variabile `lista`, dato che le virgolette non inibiscono il metacarattere `$`.

3. Si predisponga uno script della shell `ord_ext.sh` che prenda come argomento sulla linea di comando il percorso di una directory e compia le seguenti azioni:
   (a) controlli il numero degli argomenti forniti, terminando la propria esecuzione nel caso il numero sia diverso da uno;
   (b) controlli che il percorso fornito corrisponda ad una directory e sia leggibile dall'utente (altrimenti l'esecuzione deve terminare);
   (c) esegua il comando `ls` senza opzioni sul percorso passato, visualizzandone l'output ordinato in base all'**estensione** dei vari file (per semplicità, si supponga che in ogni nome di file vi sia al più un'occorrenza del carattere punto ’.’).

   Ad esempio se nella directory `/home/user/test` sono contenuti i seguenti file:

   ```
   compito.doc  fact.sh  host  manuale.pdf  mthread.c  prova.sh  relazione.tex
   ```

   il comando `./ord_ext.sh /home/user/test` deve restituire quanto segue:

```
host
mthread.c
compito.doc
manuale.pdf
fact.sh
prova.sh
relazione.tex
```

Soluzione:

Esempio di soluzione:

```
1  if test $# -ne 1
2  then
3    echo "Utilizzo dello script: $0 <path>"
4    exit 1
5  fi
6
7  if ! test -d $1 -a -r $1
8  then
9    echo "Il percorso $1 deve essere leggibile e deve essere una
         directory"
10   exit 2
11 fi
12
13 ls -r $1 | sort -t'.' -k2,3
14
15 exit 0
```

4. Sia data la seguente struttura ricorsiva in C per la rappresentazione di alberi binari:

```
struct bintree {
  int val;
  struct bintree *left;
  struct bintree *right;
};
```

dove `val` rappresenta il valore del nodo, mentre `left` e `right` puntano, rispettivamente, al figlio sinistro ed al figlio destro (tali puntatori assumono il valore `NULL` quando non esistono i rispettivi figli).

Si scriva il codice di una funzione avente il seguente prototipo:

```
int conta(struct bintree *root);
```

che restituisca come valore di ritorno il numero di nodi (elementi) dell'albero binario puntato da `root`.

Soluzione:

Esempio di soluzione:

```
1 int conta(struct bintree *root) {
2     if(root==NULL) return 0;
3     else
4         return 1+conta(root->left)+conta(root->right);
5 };
```

Laboratorio di Sistemi Operativi
14 luglio 2015
Compito

5. Il programma seguente utilizza i thread ed i relativi meccanismi di accesso esclusivo (mutex) e di sincronizzazione (condition variable) introdotti a lezione, 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)
  }
};
```

Soluzione:

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);`