Laboratorio di Sistemi Operativi
9 febbraio 2016
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Il comando `date` produce in output la data ed ora attuali: ad esempio,

   `Tue  9 Feb 10:23:04 CET 2016`

   Si scriva uno script che produca in output una stringa nel formato

   `giorno/mese/anno - ore_minuti_secondi` (ad esempio: `9/Feb/2016 - 10_23_04`).

   Soluzione:

   Esempio di soluzione:

   ```
   #!/bin/bash
   dataora=`date | tr -s ' '`
   giorno=`echo $dataora | cut -d ' ' -f2`
   mese=`echo $dataora | cut -d ' ' -f3`
   anno=`echo $dataora | cut -d ' ' -f6`
   ora=`echo $dataora | cut -d ' ' -f4 | tr ':' '_'`
   echo "$giorno/$mese/$anno - $ora"
   ```

2. Qual è l'effetto dei seguenti comandi?

   1. `ls -l | grep '^d...r.x'`
   2. `n=`wc registro.txt | sed 's/ /:/g' | cut -d':' -f4`` si supponga che l'output di `wc registro.txt` sia la stringa '␣14␣␣27␣415␣registro.txt' (senza apici, ma con gli spazi evidenziati);
   3. `echo $n` (dove `n` è la variabile inizializzata nel punto precedente).

   Attenzione: nel punto 2 gli apici esterni sono dei *backquote* (apici rovesciati).

   Soluzione:

   1. L'output di questa pipeline consiste nella visualizzazione in long format delle subdirectory della directory corrente che sono leggibili ed attraversabili dagli utenti del gruppo. Infatti il pattern usato con il comando `grep` seleziona le linee prodotte da `ls -l` che iniziano con una `d` (directory) e che hanno i permessi `r` e `x` del gruppo attivi.
   2. L'output prodotto dal primo comando della pipeline ('␣14␣␣27␣415␣registro.txt') viene passato al comando `sed` che sostituisce tutti gli spazi con dei due punti (`:`), producendo la stringa ' :14::27:415: registro .txt'. Quindi il comando `cut` seleziona il quarto campo (usando i due punti come separatore) producendo come output finale della pipeline `27`. Questo è quanto viene assegnato alla variabile `n`, dato che tutta la pipeline si trova all'interno dei backquote, inducendo la shell a compiere una *command substitution*.
   3. viene stampato il valore della variabile `n`, ovvero, `27`.

3. Sia data la seguente struttura ricorsiva in C per la rappresentazione di elementi di una coda di interi:

   ```
   struct int_queue {
     int val;
     struct int_queue *next;
   };
   ```

   dove `val` rappresenta il valore dell'elemento della lista, mentre `next` punta al prossimo elemento della lista (se l'elemento in questione è l'ultimo, allora `next` punta a `NULL`). La lista vuota è così rappresentabile da un puntatore di tipo `struct int_queue *` inizializzato a `NULL`.

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

```
void append(struct int_queue *head, int n);
```

che inserisca il valore intero `n` in fondo alla coda puntata da `head`, allocando la memoria necessaria ed aggiornando opportunamente i puntatori.

Soluzione:

Esempio di soluzione iterativa:

```
1  void append(struct int_queue *head, int n) {
2    while(head->next!=null) {
3      head=head->next;
4    }
5
6    head->next=(struct int_queue *)malloc(sizeof(struct int_queue));
7    if(head->next==NULL) {
8      perror("Errore di allocazione della memoria!\n");
9      return;
10   }
11   head->next->val=n;
12   head->next->next=NULL;
13 }
```

4. Si consideri il seguente programma `ordered_vector.c` (per semplicità sono state omesse le direttive `include`):

```
1  int *buf; /* la variabile è globale, ovvero, accessibile a tutti i thread
2               in modo concorrente! */
3  void *generate(void *n) { /* funzione che genera un elemento intero
4                               e lo inserisce in modo ordinato in buf */
5    int i,t,r;
6    r=random();              // genero il valore intero casuale
7    printf("Valore generato: %d\n",r);
8
9    for(i=0;i<((int)n);i++) {
10     if(buf[i]==-1) {       // la posizione corrente è libera
11       buf[i]=r;
12       break;
13     }
14     else {
15       if(buf[i]<=r) /* posizione occupata da un elemento minore od uguale:
                          proseguo con l'iterazione successiva */
16         continue;
17       else {
18         sleep(random()%2+1); // il thread attende da 1 a 2 secondi
19         /* posizione occupata da un elemento maggiore: scambio i valori di r e
                di buf[i] */
20         t=buf[i];
21         buf[i]=r;
22         r=t;
23       }
24     }
25   }
26 }
27
28 int main(int argc, char *argv[]) {
29   int i,n=atoi(argv[1]);
30   pthread_t *thr;
31   thr=(pthread_t *)malloc(n*sizeof(pthread_t)); /* alloco memoria per un array
          di n thread id */
```

```
32  buf=(int *)malloc(n*sizeof(int)); /* alloco memoria per un array di n interi
        */
33
34  for(i=0;i<n;i++) { /* gli elementi del vettore sono inizializzati a -1 */
35          buf[i]=-1;      /* la posizione i-esima è libera */
36  }
37
38  srandom(time(NULL)); /* inizializzo il generatore di numeri casuali */
39  for(i=1; i<=n; i++) {
40    if(pthread_create(&thr[i-1],NULL,generate,(void *)n)!=0) { // creo il
          thread i-esimo
41      perror("Errore nella creazione del thread.\n");
42      exit(1);
43    }
44  }
45
46  for(i=1;i<=n;i++) {
47    pthread_join(thr[i-1],NULL); /* attendo la terminazione dell'i-esimo
          thread figlio */
48  }
49  free(thr); // libero la memoria allocata per il vettore di thread id
50
51  for(i=0;i<n;i++) { // stampo il contenuto di buf
52        printf("buf[%d]=%d\n",i,buf[i]);
53  }
54  free(buf); // libero la memoria allocata per il vettore buf
55  return 0;
56 }
```

Una volta compilato, l'eseguibile accetta un intero `n` sulla linea di comando e genera un numero `n` di thread figli, ognuno dei quali (tramite la funzione `generate`) produce un valore intero casuale, memorizzandolo nel vettore `buf` in modo ordinato crescente:

```
> ./ordered_vector 5
Valore generato: 1807604932
Valore generato: 38608569
Valore generato: 1799968236
Valore generato: 1322540687
Valore generato: 1094827781
buf[0]=38608569
buf[1]=1094827781
buf[2]=1322540687
buf[3]=1799968236
buf[4]=1807604932
```

(a) Il programma riportato funzionerà correttamente, ovvero, produrrà in output il vettore ordinato? Perché?

(b) Se si è risposto negativamente alla precedente domanda, modificare (si può cancellare/modificare/aggiungere codice a piacimento) il programma in modo che rispetti la consegna.

Soluzione:

(a) Il programma non funzionerà correttamente in quanto i vari thread potranno accedere e modificare in modo concorrente il vettore condiviso `buf` con operazioni non atomiche. Lo stato finale del vettore sarà quindi dipendente da una corsa critica. Per evitare ciò si può far ricorso ad un mutex in questo modo:

1. dichiarare ed inizializzare un mutex globalmente accessibile, ad esempio, aggiungendo la linea seguente fra le righe 2 e 3:

```
1       pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
```

2. isolare la sezione critica della funzione `generate` introducendo in linea 8 la chiamata seguente:

```
1       pthread_mutex_lock(&mutex);
```

   ed introducendo una nuova linea fra le righe 25 e 26 con la chiamata duale:

```
1       pthread_mutex_unlock(&mutex);
```

Un'altra possibilità è quella di sequenzializzare l'esecuzione dei thread, facendo in modo che il padre attenda la terminazione del thread appena creato, prima di procedere alla creazione del successivo. Per far ciò è sufficiente eliminare il ciclo `for` da riga 46 a riga 48 e spostare l'istruzione in riga 47 in una nuova linea fra le righe 43 e 44.