Caselle riservate

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Sistemi Operativi

Compito d'esame

28 Gennaio 2016

Matricola _____ Cognome _____ Nome _____

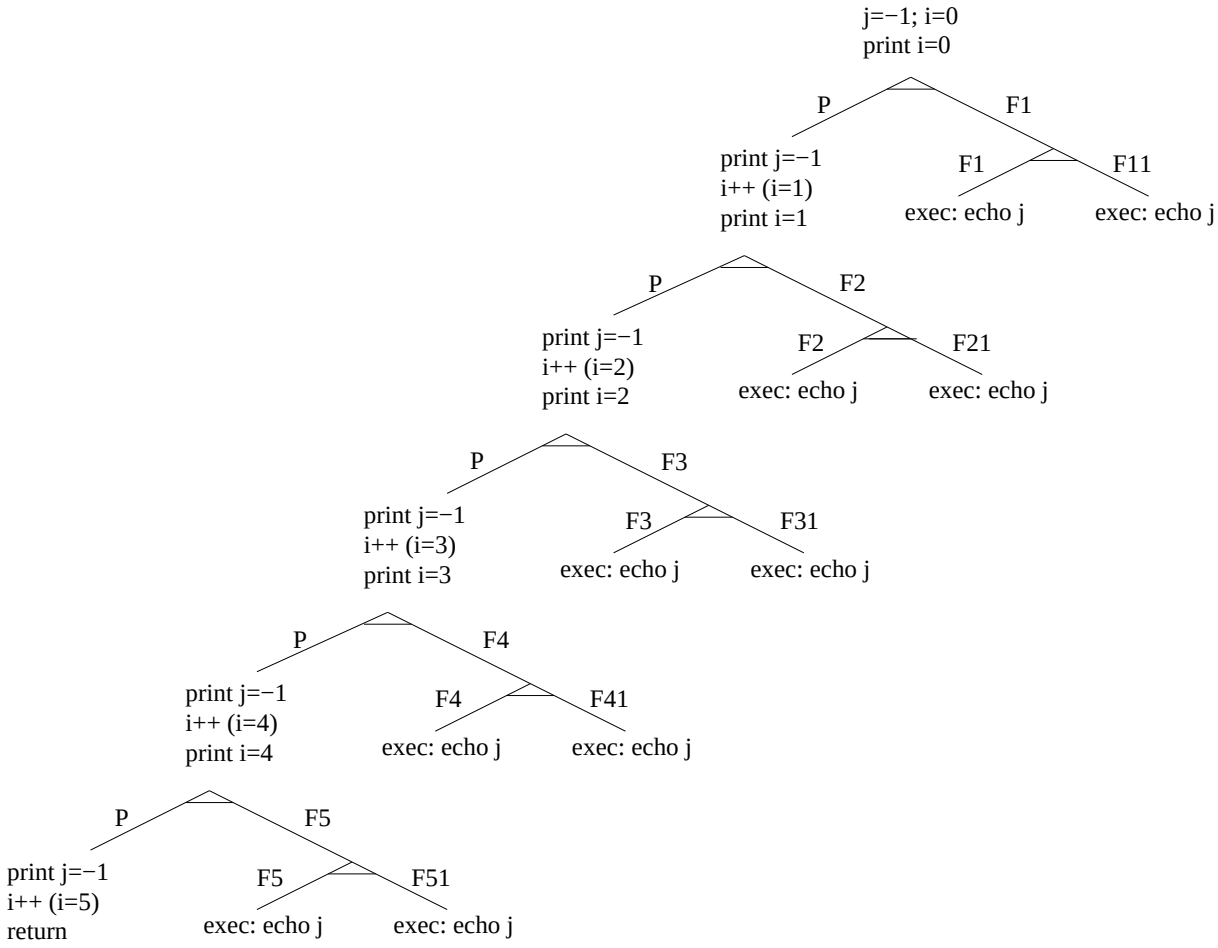
Docente: ◯ Quer ◯ Sterpone

L'unico materiale consultabile durante la prova scritta consiste nei tre formulari predisposti dal docente. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 100 **minuti.**

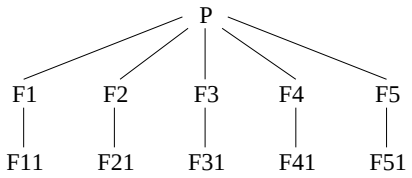
1. Si riporti il control flow graph e l'albero di generazione dei processi ottenuto dall'esecuzione del seguente tratto di codice C. Si indichi inoltre che cosa esso produce su video e per quale motivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main () {
  int i, j=-1;
  for (i=0; i<5; i++) {
    printf ("i=%d\n", i);
    if (!fork()) {
      for (j=2; j>0; j--) {
        fork();
        execlp ("echo", "i", "j", (char *) 0);
      }
    } else {
      printf ("j=%d \n", j);
    }
  }
  return (1);
}
```

CFG:



Albero di generazione dei processi:

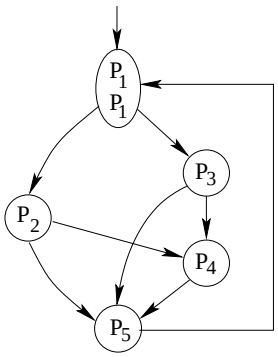


Output prodotto:

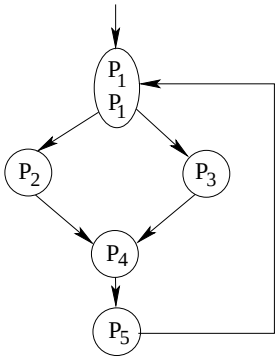
```
i=0
i=1
i=2
i=3
i=4
j=-1 compare 5 volte
j    (generato dalla echo) compare 10 volte
```

con 20 stampe in tutto.

2. Dato il seguente grafo di precedenza, realizzarlo utilizzando il **minimo** numero possibile di semafori. I processi rappresentati devono essere processi ciclici (con corpo del tipo `while(1)`). Utilizzare le primitive `init`, `signal`, `wait` e `destroy`. Indicare gli eventuali archi superflui e riportare il corpo dei processi ($P_1, \ldots, P_5$) e l'inizializzazione dei semafori. Si noti che P_1 deve effettuare due iterazioni in sequenza del proprio ciclo `while` per ogni iterazione globale sull'intero grafo di precedenza.



Gli archi P_3-P_5 e P_2-P_5 sono superflui. Il grafo di precedenza risultate è il seguente:



Ovviamente una soluzione del tipo:

```
P1
while (1) {
  wait (s1);
  printf ("P1\n");
  printf ("P1\n");
  signal (s2);
  signal (s3);
}
```

non è quanto richiesto. Quindi, sono possibili le due seguenti soluzioni.

Soluzione A

Prima dell'esecuzione:

```
sem_t s1, s2, s3, s4, s5;
init (s1, 1);
init (s2, 0);
init (s3, 0);
init (s4, 0);
init (s5, 0);
int n = 0;
```

Si esegue una istanza di ciascun processo P_i.

```
P1
while (1) {
  wait (s1);
  printf ("P1\n");
  if (n==0) {
    signal (s1);
    n=1;
  } else {
    n=0;
    signal (s2);
    signal (s3);
  }
}
```

```
P2
while (1) {
  wait (s2);
  printf ("P2\n");
  signal (s4);
}
```

```
P3
while (1) {
  wait (s3);
  printf ("P3\n");
  signal (s4);
}
```

```
P4
while (1) {
  wait (s4);
  wait (s4);
  printf ("P4\n");
  signal (s5);
}
```

```
P5
while (1) {
  wait (s5);
  printf ("P5\n");
  signal (s1);
}
```

Al terrmine dell'esecuzione:

```
destroy (s1);
...
destroy (s5);
```

Soluzione B

Prima dell'esecuzione:

```
sem_t s1, s2, s3, s4, s5;
init (s1, 2);
init (s2, 0);
init (s3, 0);
init (s4, 0);
init (s5, 0);
```

Si esegue una istanza di ciascun processo P_i.

```
P1
while (1) {
  wait (s1);
  printf ("P1\n");
  signal (s2);
  signal (s3);
}
```

```
P2
while (1) {
  wait (s2);
  wait (s2);
  printf ("P2\n");
  signal (s4);
}
```

```
P3
while (1) {
  wait (s3);
  wait (s3);
  printf ("P3\n");
  signal (s4);
}
```

```
P4
while (1) {
  wait (s4);
  wait (s4);
  printf ("P4\n");
  signal (s5);
}
```

```
P5
while (1) {
  wait (s5);
  printf ("P5\n");
  signal (s1);
  signal (s1);
}
```

Al terrmine dell'esecuzione:

```
destroy (s1);
...
destroy (s5);
```

3. Con riferimento alle Sezioni Critiche, si indichino i requisiti che ogni soluzione deve soddisfare e quali classi di soluzioni sono proponibili (con le rispettive caratteristiche e diversità).

   Si faccia quindi riferimento alla sincronizzazione di processi mediante la procedura `swap`. Se ne illustrino le principali caratteristiche, riportandone il codice e il relativo protocollo di utilizzo.

   Ogni soluzione al problema delle SC deve soddisfare i seguenti requisiti:

   - Mutua esclusione: un solo P (o T) alla volta deve ottenere laccesso alla regione critica
   - Progresso: se nessun P (o T) si trova nella SC e un P (o T) desidera entrarci, deve poterlo fare in un tempo definito
     - Solo i P (o T) in fase di prenotazione possono partecipare alla selezione
     - Nessun P (o T) fuori dalla SC pu bloccare altri P (o T)

     Ovvero occorre evitare deadlock tra P (o T)A
   - Attesa definita: deve esistere un numero definito di volte per cui altri P (o T) riescano ad accedere alla SC prima che un P (o T) specifico e che ha fatto una richiesta di accesso possa farlo. Ovvero, occorre evitare starvation di P (o T).
   - Ogni soluzione dovrebbe essere simmetrica: la selezione di chi deve accedere alla SC non dovrebbe dipendere dalla priorità relativa tra P (o T) e dalla velocitaà relativa dei P (o T).

   Le SC ammettono soluzioni:

   - Software: algoritmi la cui correttezza risiede nella logica del software.
   - Hardware: strategie che si basano su caratteristiche specifiche dellhardware, ovvero istruzioni macchina che richiedono requisiti di esecuzione particolari.
   - Ad-Hoc: il sistema operativo fornisce funzioni e strutture dati al programmatore che le utilizza in maniera. opportuna

   Soluzione hardware con `swap`. Funzione:

```
void swap (char *v1, char *v2) {
  char tmp;
  tmp = *v1;
  *v1 = *v2;
  *v2 = tmp;
  return;
}
```

   Protocollo di utilizzo (per un numero di processi qualsiasi):

```
while (TRUE) {
  key = TRUE;
  while (key==TRUE) {
    swap (&lock, &key); // Lock
  }
  SC
  lock = FALSE;
  sezione non critica
}
```

   Spiegazione ....

   Le tecniche precedenti: assicurano la mutua esclusione, assicurano il progresso, evitando il deadlock, Non assicurano lattesa definita di un processo, ovvero non garantiscono la non starvation, Sono simmetriche. Per soddisfare tutti e Quattro i criteri occorre estendere le soluzioni precedenti.

4. Scrivere uno script BASH (senza l'utilizzo di AWK) che riceve tre parametri:

   `n cmd dir`

   in cui `n` indica un valore numerico intero, `cmd` è una stringa che rappresenta un comando (singolo) di shell e `dir` è il nome di un direttorio.

   Lo script deve eseguire il comando `cmd` sui primi `n` file presenti nel direttorio `dir`. Si supponga che `n` sia sempre inferiore del numero di file presenti nel direttorio e che i file vadano selezionati in ordine di dimensione decrescente.

   Suggerimento: si ricorda che nell'output del comando "`ls -l`"

   ```
   -rw-rw-r-- 1 quer quer  42729 Jan 25 11:33 file1.txt
   -rw-rw-r-- 1 quer quer 226662 Jan 25 11:33 exam.ps
   ...
   ```

   i campi sono separati da una tabulazione e la dimensione del file è indicata nel quinto campo.

   Soluzione A:

   ```
   #!/bin/bash
   # Versione 1: selezione file mediante head
   if [ $# -lt 3 ]; then
     echo "Usage $0 n cmd dir"
     exit 0
   fi
   for file in $(ls -l $3 | tr -s " " \
     | grep -e "^-" | sort -nr -k 5   \
     | cut -d " " -f 9 | head -n $1); do
       eval "$2 $3/$file" #oppure $2 $3"/"$file
   done
   ```

   Soluzione B:

   ```
   #!/bin/bash
   # Versione 2: selezione file mediante ciclo while
   if [ $# -lt 3 ]; then
     echo "Usage $0 n cmd dir"
     exit 0
   fi
   n=$1
   cmd=$2
   dir=$3
   ls -l $3 | tr -s " " |
     \ sort -nr -k 5,5 | cut -d " " -f 9 > tmp.txt
   i=0
   while read f
   do
     if [ -f "$dir/$f" ]
     then
       $cmd $dir/$f
       i=$(($i+1))
     fi
     if [ $i -ge $n ]
     then
        break
     fi
   done < tmp.txt
   rm -f tmp.txt
   ```

   Soluzione C:

   ```
   #!/bin/bash
   # Versione 3: selezione file mediante ciclo for
   if [ $# -lt 3 ]; then
       echo "Usage $0 n cmd dir"
       exit 0
   fi
   n=$1
   cmd=$2
   dir=$3
   i=0
   for f in ‘ls -l $3 | tr -s " " | sort -nr -k 5,5 | cut -d " " -f 9‘
   do
     if [ -f "$dir/$f" ]
     then
       $cmd $dir/$f
       i=$(($i+1))
     fi
     if [ $i -ge $n ]
     then
        break
     fi
   done
   ```

5. La base dati relativa a un esame è costituita da due file.

   Il primo file riporta le generalità degli studenti, con formato:

   `cognome nome numeroMatricola`

   Il secondo file memorizza il risultato ottenuto da tutti gli studenti in tutti gli esami superati:

   `numeroMatricola nomeEsame voto`

   Scrivere un script AWK in grado di ricevere sulla riga di comando 5 parametri:

   `nomeFile1 nomeFile2 numeroMatricola x sogliaVoto`

   dove `nomeFile1` e `nomeFile2` sono file con il formato precedentemente descritto e `x` è un operatore relazionale, ovvero una tra le stringhe (`<`, `==`, `>`). Lo script deve visualizzare (a video) l'elenco di tutti gli esami sostenuti dallo studente di matricola indicata che soddisfano la condizione relazionale "`voto x sogliaVoto`". Il formato di uscita deve essere del tipo:

   `cognome nome nomeEsame voto`

```
#!/usr/bin/awk
# Esempio chiamata da linea di comando:
# awk -f awk_esame28.awk -v m=1 x=">" v=19 file2="matricola_voto.txt"  nome_matricola.txt
{
  matricola=$3
  vett_nome[matricola]=$1
  vett_cognome[matricola]=$2
}
END{
  while(getline < file2) {
    if($1 == m) {
      voto=$3
      if (( x ~"^>$" &&voto > v) || (x~"^==$"&&voto == v) ||
          ( x~"^<$" && voto < v )) {
         print vett_nome[$1] " " vett_cognome[$1] " " $2 " " $3
      }
    }
  }
}
```

6. Si consideri il seguente insieme di processi:

Processo	Tempo arrivo	Burst Time	Priorità
P_1	0	2	5
P_2	1	8	1
P_3	2	1	4
P_4	3	6	2
P_5	4	4	3

Rappresentare mediante diagramma di Gantt l'esecuzione di tali processi utilizzando gli algoritmi di scheduling PS (Priority Scheduling), RR (Round Robin) e SRTF (Shortest Remaining Time First).

La priorità maggiore è associata al valore di priorità inferiore. Per la strategia Round Robin si utilizzi un quantum temporale di 2 unità di tempo.