Caselle riservate

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

Sistemi Operativi

Compito d'esame

28 Gennaio 2013

Versione A

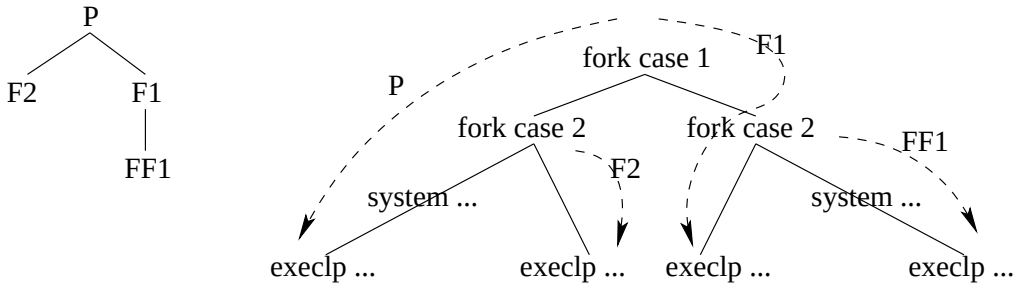
Matricola _____ Cognome _____ Nome _____

Docente: ◯ Laface ◯ Quer

Non si possono consultare testi, appunti o calcolatrici. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 60 **minuti.**

1. Si riporti l'albero di generazione dei processi e si indichi che cosa produce su video il seguente programma e per quale motivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
  pid_t pid;
  int i;
  for (i=1; i<=3; i++){
    switch (i) {
      case 1: fork(); break;
      case 2: pid=fork(); if (pid!=0) system ("echo case 2"); break;
      case 3: execlp ("echo", "myPgrm", "case 3", NULL); break;
    }
  }
  return (0);
}
```



L'output ottenuto è il seguente:

```
case 2
case 2
case 3
case 3
case 3
case 3
```

Veloce descrizione di fork, system e execlp. L'ordine dell'output dipende dallo scheduler.

2. Si illustri il problema dei *Readers e Writers* riportandone la soluzione per il caso di precedenza ai Readers mediante semafori. Si indichi la funzione dei vari semafori motivandone l'utilizzo.

Semafori e variabili globali:

```
nr = 0;
init (w, 1);
init (meR, 1);
```

Readers:

```
wait (meR);
  nr++;
  if (nr==1)
    wait (w);
signal (meR);
...
lettura
...
wait (meR);
  nr--;
  if (nr==0)
    signal (w);
signal (meR);
```

Writers:

```
wait (w);
...
scrittura
...
signal (w);
```

Il semaforo `w` serve per realizzare la mutua esclusione tra Readers e Writes o tra diversi writers.

Il semaforo `meR` serve per realizzare la mutua esclusione tra Readers nella fase di modifica di `nr`.

`nr` conteggia il numero di Readers nella sezione critica.

3. Si illustri l'*algoritmo del banchiere*. Analizzando l'esempio successivo (con processi $(P_0, \ldots, P_4)$ e risorsa R) si indichi se lo stato è sicuro o non sicuro e si riporti la sequenza sicura o non sicura.

Processo	Fine	Assegnate R	Massimo R	Necessità R	Disponibilità R
P_0	No	2	7		2
P_1	No	2	3		
P_2	No	2	8		
P_3	No	0	3		
P_4	No	1	5		

L'algoritmo del banchiere serve per evitare il deaclock nel caso di risorse con instanze multiple (altrimenti è sufficiente un algoritmo di determinazioe dei cicli sul grafo di assegnazione).

Lo stato dell'esempio è sicuro.

Sequenza sicura:
P_1 (disponibili=4)
P_3 (disponibili=4)
P_4 (disponibili=5)
P_0 (disponibili=7)
P_2 (disponibili=9).

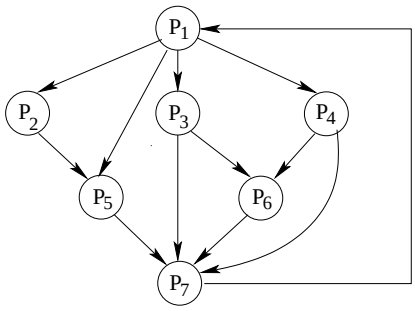
Verifica di una richiesta:

```
se
  per ogni j Richieste[i][j]<=Necessit[i][j]
  AND
  per ogni j Richieste[i][j]<=Disponibili[j]
  ALLORA
    per ogni j Disponibili[j]=Disponibili[j]-Richieste[i][j]
    per ogni j Assegnate[i][j]=Assegnate[i][j]+Richieste[i][j]
    per ogni j Necessit[i][j]=Necessit[i][j]-Richieste[i][j]
```

Verifica di uno stato:

```
1.
Inizializza
Fine [i] = falso per tutti i Pi
2.
Trova un Pi per cui
Fine[i]=falso && per ogni j Necessit[i][j]<=Disponibili[j]
Se tale i non esiste vai al passo 4
3.
per ogni j Disponibili[j]=Disponibili[j]+Assegnate[i][j]
Fine[i] = vero
Vai al passo 2
4.
Se Fine[i]=vero per tutti i Pi
Allora il sistema  in uno stato sicuro
```

4. 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)`). Si utilizzino le primitive `init`, `signal` e `wait`. Riportare il corpo dei processi $(P_1, \ldots, P_7)$ e l'inizializzazione dei semafori.



Gli archi P_1–P_5, P_3–P_7 e P_4–P_7 sono superflui.

Prima dell'esecuzione:

```
sem_t s1, s2, s3, s4, s5, s6, s7;

init (s1, 1);
init (s2, 0);
..
init (s7, 0);
```

Al termine dell'esecuzione:

```
destroy (s1);
..
destroy (s7);
```

P1

```
while (1) {
  wait (s1);
  printf ("P1\n");
  signal (s2);
  signal (s3);
  signal (s4);
  signal (s5); // Superfluo
}
```

P2

```
while (1) {
  wait (s2);
  printf ("P2\n");
  signal (s5);
}
```

P3

```
while (1) {
  wait (s3);
  printf ("P3\n");
  signal (s6);
  signal (s7); // Superfluo
}
```

P4

```
while (1) {
  wait (s4);
  printf ("P1\n");
  signal (s6);
  signal (s7); // Superfluo
}
```

P5

```
while (1) {
  wait (s5);
  wait (s5); // Superfluo
  printf ("P5\n");
  signal (s7);
}
```

P6

```
while (1) {
  wait (s6);
  wait (s6);
  printf ("P6\n");
  signal (s7);
}
```

P7

```
while (1) {
  wait (s7);
  wait (s7);
  wait (s7); // Superfluo
  wait (s7); //Superfluo
  printf ("P7\n");
  signal (s1);
}
```

5. Realizzare uno script bash che riceva come unico argomento un file di testo. Lo script deve:
   - effettuare una copia del file in un file con lo stesso nome ma con estensione `xyx`
   - modificare il file originario come segue:
     - aggiungere all'inizio di ogni riga il numero di parole della riga e il numero di righe totali del file
     - ordinare le righe in ordine crescente in base al numero di parole.

   Non si ricorra all'utilizzo di AWK.

```
#!/bin/bash
if [ $# -ne 1 ]
then
  echo "usage $0 file.txt"
  exit 1;
fi
newfilename=‘basename $1 "*.txt"‘
newfilename=$newfilename".xyz"
cat $1 > $newfilename # or cp $1 $newfilename
nlines=‘cat $1|wc -l‘
rm -f tmp1.txt
while read line
do
  nwords=‘echo $line | wc -w‘
  echo $nwords $nlines $line  >> tmp1.txt
done < $1
cat tmp1.txt | sort -k 1 -n > $1
rm tmp1.txt
```

6. Un file contiene un testo di lunghezza indefinita ma senza caratteri di interpunzione. Scrivere uno script AWK che, ricevuto il nome di tale file sulla riga di comando, visualizzi su standard output l'istogramma a barre del numero di occorrenze di tutte le stringhe presenti nel file di lunghezza esattamente uguale a 5 caratteri e contenenti almeno due vocali qualsiasi tra 'a', 'e', 'i', 'o', 'u'.

Esempio

File di ingresso	Output prodotto
testo di esempio che contiene molte parole	`testo` ###
con 5 caratteri e almeno 2 vocali testo	`molte` ####
barre molte molte molte barre di testo	`barre` ##

```
#!/usr/bin/awk -f
BEGIN {
  nwords=1
}
{
  for(i=1;i<=NF;i++) {
    if (length($i)==5)  {
      found=0;
      if (match($i,".*a.*")>0) {
       found++;
      }
      if (match($i,".*e.*")>0) {
       found++;
      }
      if (match($i,".*i.*")>0) {
       found++;
      }
      if (match($i,".*o.*")>0) {
       found++;
      }
      if (match($i,".*u.*")>0) {
       found++;
      }
      if (found>=2) {
        vett[$i]++;
        if ( vett[$i]==1 ) {
          words[nwords]=$i;
          nwords++;
        }
      }
    }
  }
}
END {
  for(i in words) {
    printf "%s ", words[i];
    for (j=vett[words[i]];j>=1;j--) {
      printf "# ";
    }
    printf "\n";
  }
}
```

Caselle riservate

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

Sistemi Operativi

Compito d'esame

28 Gennaio 2013

Versione B

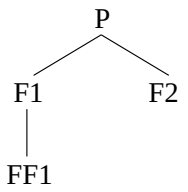
Matricola _____ Cognome _____ Nome _____

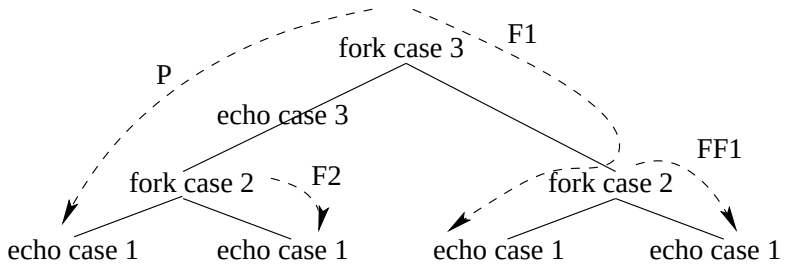
Docente: ◯ Laface ◯ Quer

Non si possono consultare testi, appunti o calcolatrici. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 60 **minuti.**

1. Si riporti l'albero di generazione dei processi e si indichi che cosa produce su video il seguente programma e per quale motivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
  pid_t pid;
  int i;
  for (i=3; i>=1; i--){
    switch (i) {
      case 1: execlp ("echo", "myPgrm", "case 1", NULL); break;
      case 2: fork(); break;
      case 3: pid=fork(); if (pid!=0) system ("echo case 3"); break;
    }
  }
  return (0);
}
```






```
Output:
case 3
case 1
case 1
case 1
case 1
```

Veloce descrizione di fork, system e execlp. L'ordine dell'output dipende dallo scheduler.

2. Si illustri il problema dei *Produttore e Consumatore* riportandone la soluzione per il caso di un solo produttore e di tre consumatori. Si indichi la funzione dei vari semafori motivandone l'utilizzo.

Semafori e variabili globali:

```
init (full, 0);
init (empty, MAX);
init (MEc, 1);
```

Produttore:

```
Producer () {
  Message m;
  while (TRUE) {
    produce m;
    wait (empty);
    enqueue (m);
    signal (full);
  }
}
```

Consumatore:

```
Consumer () {
  Message m;
  while (TRUE) {
    wait (full);
    wait (MEc);
    m = dequeue ();
    signal (MEc);
    signal (empty);
    consuma m;
  }
}
```

Il semaforo `empty` conteggia il numero di elementi vuoti e blocca il produttore nel caso il buffer sia pieno.

Il semaforo `sull` conteggia il numero di elementi pieni e blocca il consumatore nel caso il buffer sia vuoto.

Il semaforo `meC` forza la mutua esclusione tra diversi consumatori.

Sui produttori il semaforo non serve, essendoci un solo produttore.

3. Si illustri l'*algoritmo del banchiere*. Analizzando l'esempio successivo (con processi $(P_0, \ldots, P_4)$ e risorsa R) si indichi se lo stato è sicuro o non sicuro e si riporti la sequenza sicura o non sicura.

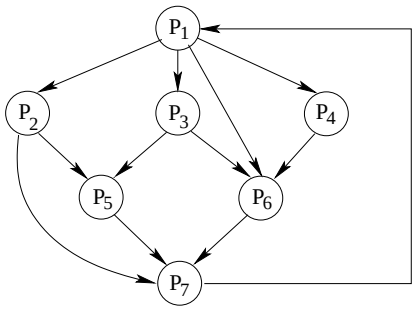
Processo	Fine	Assegnate R	Massimo R	Necessità R	Disponibilità R
P_0	No	2	11		2
P_1	No	2	3		
P_2	No	3	8		
P_3	No	0	3		
P_4	No	1	5		

Vedere decrizione e commenti nella risposta alla *Versione A*.

Lo stato dell'esempio non è sicuro.

Sequenza:
P_1 (disponibili=4)
P_3 (disponibili=4)
P_4 (disponibili=5)
P_2 (disponibili=8)
P_0 NON eseguibile.

4. 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)`). Si utilizzino le primitive `init`, `signal` e `wait`. Riportare il corpo dei processi $(P_1, \ldots, P_7)$ e l'inizializzazione dei semafori.



Gli archi P_1–P_6 e P_2–P_7 sono superflui.

```
Prima dell'esecuzione:
sem_t s1, s2, s3, s4, s5, s6, s7;

init (s1, 1);
init (s2, 0);
...
init (s7, 0);
```

```
Al termine dell'esecuzione:
destroy (s1);
...
destroy (s7);
```

```
P1
while (1) {
  wait (s1);
  printf ("P1\n");
  signal (s2);
  signal (s3);
  signal (s4);
  signal (s6); // Superfluo
}
```

```
P2
while (1) {
  wait (s2);
  printf ("P2\n");
  signal (s5);
  signal (s7);  //Superfluo
}
```

```
P3
while (1) {
  wait (s3);
  printf ("P3\n");
  signal (s5);
  signal (s6);
}
```

```
P4
while (1) {
  wait (s4);
  printf ("P1\n");
  signal (s6);
}
```

```
P5
while (1) {
  wait (s5);
  wait (s5);
  printf ("P5\n");
  signal (s7);
}
```

```
P6
while (1) {
  wait (s6);
  wait (s6);
  wait (s6);  // Superfluo
  printf ("P6\n");
  signal (s7);
}
```

```
P7
while (1) {
  wait (s7);
  wait (s7);
  wait (s7);  //Superfluo
  printf ("P7\n");
  signal (s1);
}
```

5. Uno script bash riceve sulla riga di comando il nome di tre direttori. Lo script deve visualizzare (a video) l'elenco dei nomi dei file contenuti nel primo direttorio che contengono la stringa `main` e l'elenco dei file che non la contengono. Inoltre deve copiare il primo insieme di file nel secondo direttorio e il secondo insieme di file nel terzo direttorio.

   Se il secondo e il terzo direttorio non esistono, lo script deve crearli; in caso contrario deve cancellare tutti i file in essi contenuti prima dell'esecuzione dello script. Lo script controlli inoltre il corretto passaggio dei parametri.

   Non si ricorra all'utilizzo di AWK.

```
#!/bin/bash
if [ $# -ne 3 ]
then
  echo "usage $0 dir1 dir2 dir3"
  exit 1;
fi
#check id dir1($2) dir2($3) exists
if [ ! -d $2 ] # -d better than -e
then
  mkdir $2
else
  rm $2/*
fi
if [ ! -d $3 ]
then
  mkdir $3
else
  rm $3/*
fi
echo "files that contain main string"
for file in ‘find $1 -type f‘
do
  found=‘cat $file | grep "main" |wc -w‘
  if [ $found -ne 0 ]
  then
    echo $file
    cp $file $2
  fi
done
echo "files that don’t contain main string"
for file in ‘find $1 -type f‘
do
  found=‘cat $file | grep "main" |wc -w‘
  if [ $found -eq 0 ]
  then
    echo $file
    cp $file $3
  fi
done
```

6. Due file di testo `a.txt` e `b.txt` dovrebbero contenere le stesse parole anche se non nello stesso ordine. Implementare uno script AWK che verifichi se tutte le parole presenti nel primo file sono presenti anche nel secondo file con lo stesso numero di occorrenze. Visualizzare le parole che non rispettano questa condizione.

```
#!/usr/bin/awk -f
{
  for (j=1;j<=NF;j++) {
   vett_count[$j]++;
  }
}
END{
  while ((getline < "b.txt")) {
    for (j=1; j<=NF; j++) {
      vett_count[$j]--;
    }
  }
  close("b.txt");
  for (i in vett_count) {
    if (vett_count[i]!=0)
      print i;
  }
}
```