Caselle riservate

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

Sistemi Operativi

Compito d'esame

08 Settembre 2014

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: 75 **minuti.**

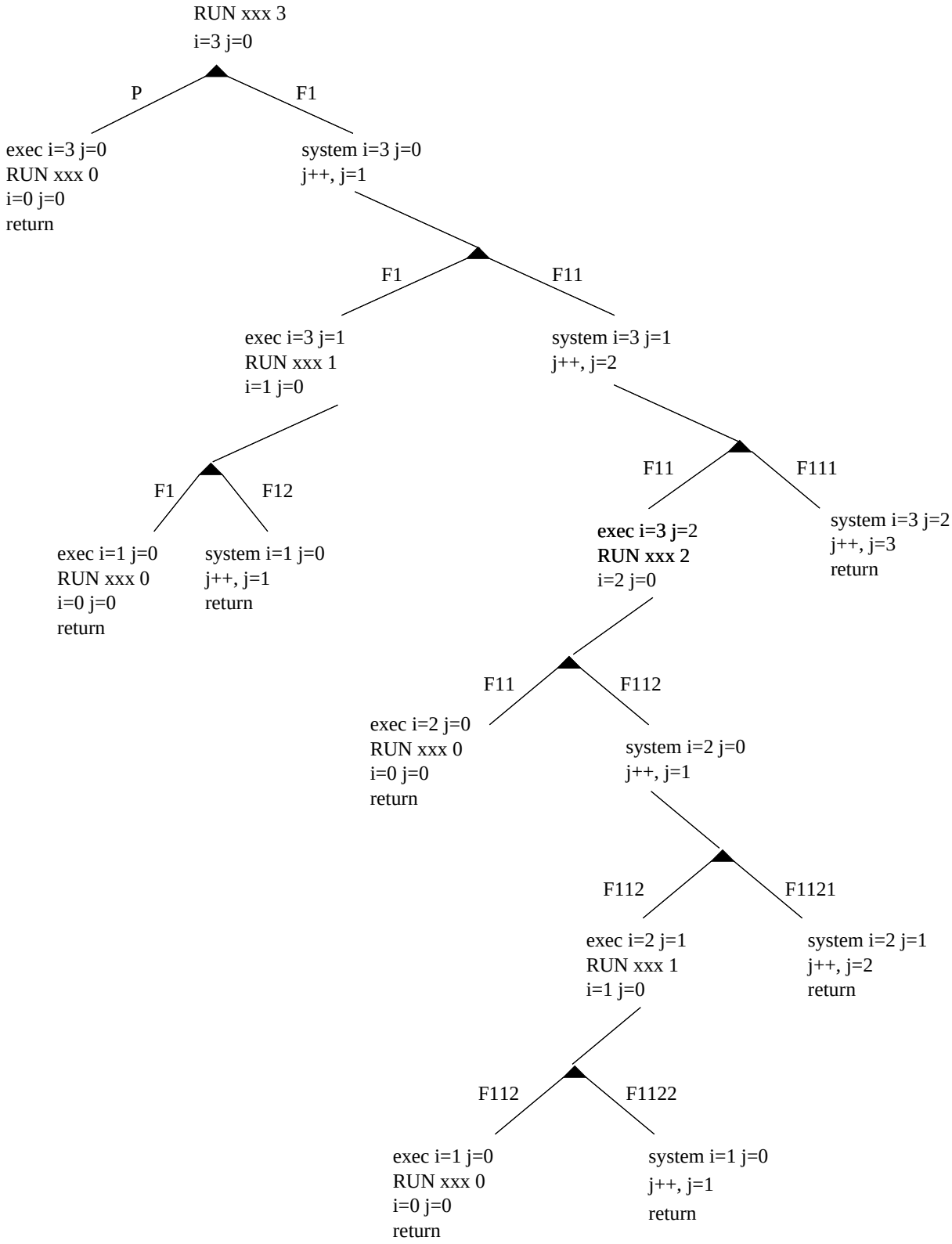
1. Si supponga che il seguente programma venga eseguito mediante l'istruzione `./pgrm xxx 3`. Si riporti l'albero di generazione dei processi a seguito della sua esecuzione. Si indichi inoltre che cosa esso produce su video e per quale motivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#define L 100
int main (int argc, char *argv[]) {
  int i, j;
  char str1[L], str2[L];
  i = atoi (argv[2]);
  for (j=0; j<i; j++) {
    if (fork () == 0) {
      sprintf (str1, "echo system with j=%d", j);
      system (str1);
    } else {
      sprintf (str1, "%s", argv[1]);
      sprintf (str2, "%d", j);
      printf ("exec ./pgrm %s %s\n", str1, str2);
      execlp ("./pgrm", "myPgrm", str1, str2, NULL);
    }
  }
  return (0);
}
```

Output prodotto:

```
exec ./pgrm xxx 0
system with j=0
exec ./pgrm xxx 1
exec ./pgrm xxx 0
system with j=1
exec ./pgrm xxx 2
system with j=0
exec ./pgrm xxx 0
system with j=2
system with j=0
exec ./pgrm xxx 1
exec ./pgrm xxx 0
system with j=1
system with j=0
```

Commenti sull'ordine delle righe in output ...


RUN xxx 3
i=3 j=0
P
F1
exec i=3 j=0
RUN xxx 0
i=0 j=0
return
system i=3 j=0
j++, j=1
F1
F11
exec i=3 j=1
RUN xxx 1
i=1 j=0
system i=3 j=1
j++, j=2
F1
F12
F11
F111
exec i=1 j=0
RUN xxx 0
i=0 j=0
return
system i=1 j=0
j++, j=1
return
exec i=3 j=2
RUN xxx 2
i=2 j=0
system i=3 j=2
j++, j=3
return
F11
F112
exec i=2 j=0
RUN xxx 0
i=0 j=0
return
system i=2 j=0
j++, j=1
F112
F1121
exec i=2 j=1
RUN xxx 1
i=1 j=0
system i=2 j=1
j++, j=2
return
F112
F1122
exec i=1 j=0
RUN xxx 0
i=0 j=0
return
system i=1 j=0
j++, j=1
return

2. Si descriva il comportamento di una shell per effettuare l'esecuzione di comandi in foreground e in background. Si riporti il codice/pseudo-codice corrispondente illustrandone il funzionamento.

   Discussione sulla system call `exec`.

   Comandi eseguiti in background:

```
while (TRUE) {
  write_prompt;
  read_command (command, parameters);
  if (fork() == 0)
    /* Child: Execute Command */
    execve (command, parameters);
else
    /* Father: Wait Child */
    wait (&status);
}
```

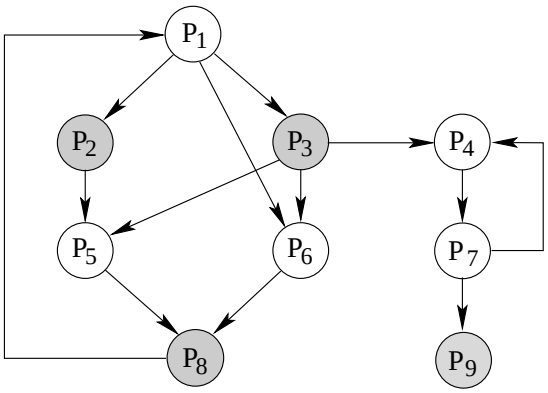
   Comandi eseguiti in foreground:

```
while (TRUE) {
  write_prompt;
  read_command (command, parameters);
  if (fork() == 0)
    /* Child: Execute Command */
    execve (command, parameters);
/* else */
    /* Padre: NON Attende */
    /* wait (&status); */
}
```

   Spiegazione e commenti ...

3. Dato il seguente grafo di precedenza, realizzarlo utilizzando il **minimo** numero possibile di semafori. I processi bianchi sono processi aciclici mentre quelli grigi sono ciclici (con corpo del tipo `while(1)`).

   Si utilizzino le primitive `init`, `signal`, `wait` e `destroy`. Riportare l'inizializzazione e la distruzione dei semafori e il corpo dei processi ($P_1, \ldots, P_9$).



L'arco P_1–P_6 è superfluo.

```
Prima dell'esecuzione:
sem_t s2, s3, s4, s5, s8, s9;
...
init (s2, 0);
init (s3, 0);
init (s4, 1);
init (s5, 0);
init (s8, 0);
init (s9, 0);
```

```
Al termine dell'esecuzione:
destroy (s2);
...
destroy (s9);
```

e si eseguono P1(), P2(), P3(), P8() e P9().

```
P1
printf ("P1\n");
signal (s2);
signal (s3);
exit (0);
```

```
P2
while (1) {
  wait (s2);
  printf ("P2\n");
  if (fork()==0) {
    P5();
  }
}
```

```
P3
while (1) {
  wait (s3);
  printf ("P3\n");
  signal (s5);
  if (fork()==0) {
    P4();
  }
  if (fork()==0) {
    P6();
  }
}
```

```
P4
wait (s4);
printf ("P4\n");
if (fork()==0) {
  P7();
}
exit (0);
```

```
P5
wait (s5);
printf ("P5\n");
signal (s8);
exit (0);
```

```
P6
printf ("P6\n");
signal (s8);
exit (0);
```

```
P7
printf ("P7\n");
signal (s4);
signal (s9);
exit (0);
```

```
P8
while (1) {
  wait (s8);
  wait (s8);
  printf ("P8\n");
  if (fork()==0) {
    P1();
  }
}
```

```
P9
while (1) {
  wait (s9);
  printf ("P9\n");
}
```

4. Si illustri il problema dei *Readers e Writers* riportandone la soluzione per il caso di precedenza ai Writers mediante semafori. Si indichi la funzione dei vari semafori motivandone l'utilizzo. Che cosa si intende per "precedenza ai Writers"?

   Vedere lucidi e relative spiegazioni oppure i testi consigliati.

5. Il comando ‘‘ls -la’’ fornisce un output simile al successivo:

```
drwxrwxr-x  9 quer quer  4096 Jan  6 13:54 current
drwxr-xr-x  2 quer quer  4096 Feb 18  2014 documents
-rw-r--r--  1 quer quer  2022 Feb 19  2014 .emacs
-rw-r--r--  1 quer quer  8445 Feb 18  2014 examples.desktop
lrwxrwxrwx  1 quer quer    10 Jan 20 10:58 so -> current/so
```

Scrivere uno script bash in grado di ricevere un elenco di nomi di direttori sulla riga di comando e per ciascuno di essi, data la lista del contenuto con formato simili a quello indicato, effettui le seguenti operazioni:

- se una directory entry è un direttorio, stampi il suo nome e il numero di sotto-direttori che tale direttorio contiene (in maniera ricorsiva).
- se una directory entry è un file regolare, stampi il nome del proprietario, la sua dimensione e il suo nome.
- se una directory entry è un link simbolico, stampi la data di creazione, il suo nome e il nome riferito dal link (senza i simboli ‘‘->’’.

```
ls -la $1 > .tmp_out
while read line
do
  f1=‘echo $line | cut -d " " -f 1‘
  r1=‘echo $f1 | grep "^d"‘ >.out_trash
  if [ $? -eq 0 ]
  then
    dname=‘echo $line | cut -d " " -f 9‘
    nsubdirs=‘find $dirname -type d | wc -l‘
    echo $dname is a dyrectory having $nsubdirs subdirs
  fi
  echo $f1 | grep "^-" >.out_trash
  if [ $? -eq 0 ]
  then
    name=‘echo $line | cut -d " " -f 9‘
    owner=‘echo $line | cut -d " " -f 3‘
    size=‘echo $line | cut -d " " -f 5‘
    echo $name is regular, its owner is $owner and its size is $size bytes
  fi
  echo $f1 | grep "^l" >.out_trash
  if [ $? -eq 0 ]
  then
    name=‘echo $line | cut -d " " -f 9‘
    date=‘echo $line | cut -d " " -f 6,7,8‘
    reference=‘echo $line | cut -d " " -f 11‘
    echo $name is a link created on $date, it refers to $reference
  fi
done < .tmp_out
```

6. Un file ASCII contiene delle valutazioni sul tempo trascorso. Ogni indicazione del tempo trascorso è indicata da due campi nella forma ‘‘n str’’, in cui n è un valore reale e str un stringa. Il valore reale n indica il tempo trascorso. La stringa str l'unità di tempo: secondo (secondi), minuto (minuti), ora (ore), giorno (giorni), mese (mesi), anno (anni). Il seguente testo riporta un esempio corretto del formato indicato:

```
1 min e’ costituito da 60 sec .
1 h e’ formata da 60 min .
2 d equivalgono a 48 h .
1 m puo’ essere costituito da 28 d , 29 d , 30 d
oppure 31 d .
```

Si osservi che ciascun valore e la corrispondente unità di misura sono campi contigui e sempre sulla stessa riga.

Si scriva uno script AWK in grado di fornire un rapporto in cui si indicano il numero di valutazioni temporali per ogni unità di grandezza e, per ciascuna di esse, la somma totale della valutazioni.

Nell'esempio precedente il rapporto dovrebbe essere del seguente tipo:

```
sec: 1 valutazione/i; somma 60
min: 1 valutazione/i; somma 61
h  : 2 valutazione/i; somma 49
d  : 5 valutazione/i; somma 120
m  : 1 valutazione/i; somma 1
y  : 0 valutazione/i; somma 0
```

Soluzione 1:

```
#!/usr/bin/awk -f
BEGIN {
  val["sec"] = 0;
  val["min"] = 0;
  val["h"]   = 0;
  val["d"]   = 0;
  val["m"]   = 0;
  val["y"]   = 0;
  sum["sec"] = 0;
  sum["min"] = 0;
  sum["h"]   = 0;
  sum["d"]   = 0;
  sum["m"]   = 0;
  sum["y"]   = 0;
}
{
  for (i=1; i<=NF; i++) {
    #print "--" $i
    if ($i=="sec" || $i=="min" || $i=="h" ||
        $i=="d" || $i=="m" || $i=="y" ) {
      val[$i]++;
      sum[$i] = sum[$i] + $(i-1);
    }
  }
}
END {
  for (i in val) {
    print i ": " val[i] " valutazione/i; somma " sum[i]
  }
}
```

Soluzione 2:

```
#!/usr/bin/awk
BEGIN {
  vals["min"] = 0
  vals["sec"] = 0
  vals["d"] = 0
  vals["h"] = 0
  vals["m"] = 0
  vals["y"] = 0
  tot["min"] = 0
  tot["sec"] = 0
  tot["d"] = 0
  tot["h"] = 0
  tot["m"] = 0
  tot["y"] = 0
}
{
  for(i=1; i<NF; i++){
    if(match($i, /^[0-9]+$/)){
      if(match($(i+1), /^min|^sec|^h|^d|^m|^y/)){
        vals[$(i+1)] = vals[$(i+1)]+1
        tot[$(i+1)] = tot["min"] + $i
      } else if(match($(i+1), /^sec/)){
        vals["sec"] = vals["sec"]+1
        tot["sec"] = tot["sec"] + $i
      } else if(match($(i+1), /^h/)){
        vals["h"] = vals["h"]+1
        tot["h"] = tot["h"] + $i
      } else if(match($(i+1), /^d/)){
        vals["d"] = vals["d"]+1
        tot["d"] = tot["d"] + $i
      } else if(match($(i+1), /^m/)){
        vals["m"] = vals["m"]+1
        tot["m"] = tot["m"] + $i
      }  else if(match($(i+1), /^y/)){
        vals["y"] = vals["y"]+1
        tot["y"] = tot["y"] + $i
      }
    }
  }
}
END {
  printf("sec: %d valutazione/i; somma %d\n", vals["sec"], tot["sec"])
  printf("min: %d valutazione/i; somma %d\n", vals["min"], tot["min"])
  printf("h: %d valutazione/i; somma %d\n", vals["h"], tot["h"])
  printf("d: %d valutazione/i; somma %d\n", vals["d"], tot["d"])
  printf("m: %d valutazione/i; somma %d\n", vals["m"], tot["m"])
  printf("y: %d valutazione/i; somma %d\n", vals["y"], tot["y"])
}
```