Caselle riservate

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

Sistemi Operativi

Compito d'esame

03 Febbraio 2014

Matricola _____ Cognome _____ Nome _____

Docente: ◯ Laface ◯ Quer

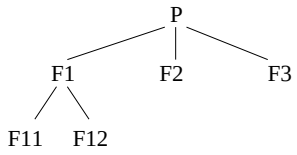
Non si possono consultare testi, appunti o calcolatrici. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 75 **minuti.**

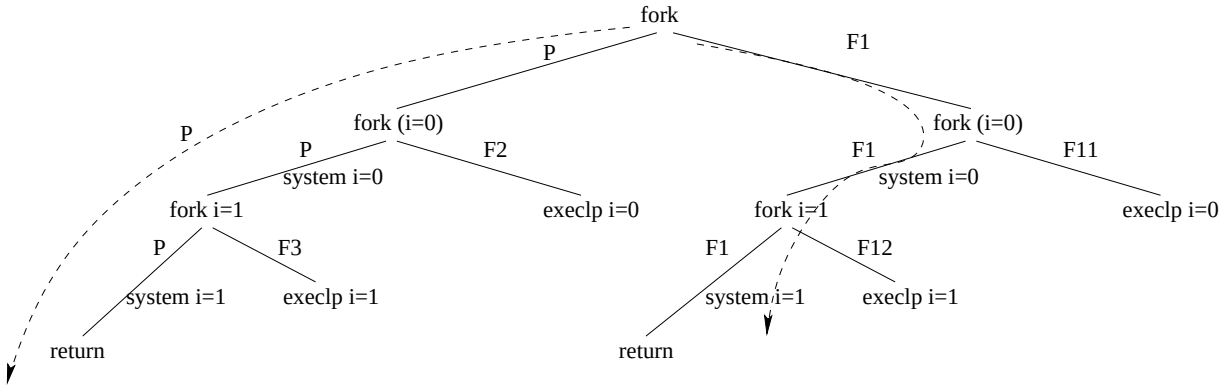
1. Chiarire il significato dei seguenti termini: *directory entry*, *symbolic-link*, *hard-link*, *soft-link*. Riportare i comandi per creare hard- e soft-link. Riportare un esempio di gestione e di conteggio degli hard-link nel caso della creazione di un direttorio quale sotto-direttorio di un direttorio dato. Rappresentare la filosofia generale e la struttura del file-system mediante ausili grafici opportuni.

   Vedere lucidi e relative spiegazioni oppure i testi consigliati.

2. Si riporti l'albero di generazione dei processi a seguito dell'esecuzione del seguente tratto di codice C. Si indichi inoltre che cosa esso produce su video e per quale motivo.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main () {
  char str[100];
  int i;
  fork();
  for (i=0; i<2; i++){
    if (fork()!=0) {
      sprintf (str, "echo system with i=%d", i);
      system (str);
    } else {
      sprintf (str, "exec with i=%d", i);
      execlp ("echo", "myPgrm", str, NULL);
    }
  }
  return (0);
}
```





Output prodotto:

```
system with i=0
system with i=0
system with i=1
system with i=1
exec with i=0
exec with i=0
exec with i=1
exec with i=1
```

Con le righe `system with i=0` prima delle righe `system with i=1` e `exec with i=1` e le righe `exec with i=0` in concorrenza con tutte le altre.

3. Si illustrino le caratteristiche delle *pipe* per la comunicazione e la sincronizzazione tra processi. Se ne illustri l'utilizzo e se ne riportino due esempi. Il primo atto a mostrarne l'utilizzo quale mezzo di comunicazione; il secondo come mezzo di sincronizzazione. Si descrivano tali esempi utilizzando tratti di codice in linguaggio C.

Una pipe permette di stabilire un flusso dati tra due processi. Ciascun processo, attraverso un file descriptor, accede a uno degli estremi della pipe. Può essere utilizzata per la comunicazione tra processi con un parente comune. Il flusso di dati è half-duplex, i.e., i dati fluiscono solo in una direzione. Lettura e scrittura da e su pipe vengono effettuate mediante `read` e `write`. Esse sono bloccanti per pipe vuota o piena, rispettivamente.

Il seguente è un esempio di utilizzo delle pipe per la realizzazione di un semaforo.

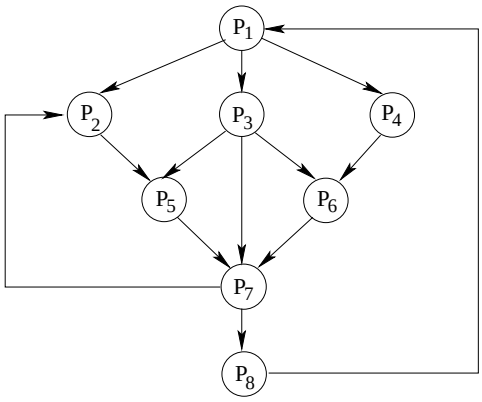
```
#include <unistd.h>
void semaphoreInit (int *S) {
  if (pipe (S) == -1) {
    printf ("Error");
    exit (-1);
  }
  return;
}
void semaphoreSignal (int *S) {
  char ctr = 'X';
  if (write(S[1], &ctr, sizeof(char)) != 1) {
    printf ("Error");
    exit (-1);
  }
  return;
}
void semaphoreWait (int *S) {
  char ctr;
  if (read (S[0], &ctr, sizeof(char)) != 1) {
    printf ("Error");
    exit (-1);
  }
  return;
}
int main() {
  int S[2];
  pid_t pid;
  semaphoreInit (S);
  pid = fork();
  if (pid == 0) {
    semaphoreWait (S);
    printf("Wait done.\n");
  } else {
    printf("Sleep 3s.\n");
    sleep (3);
    semaphoreSignal (S);
    printf("Signal done.\n");
  }
  return 0;
}
```

Mentre nel codice seguente (simile) una pipe viene utilizzata per la comunicazione.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main() {
  int n;
  int file[2];
  char cR = 'X';
  char cW;
  pid_t pid;
  if (pipe(file) == 0) {
    pid = fork ();
    if (pid == -1) {
      fprintf(stderr, "Fork failure");
      exit(EXIT_FAILURE);
    }
    if (pid == 0) {
      // child reads
      close (file[1]);
      n = read (file[0], &cR, 1);
      printf("Read %d bytes: %c\n", n, cR);
      exit(EXIT_SUCCESS);
    } else {
      // parent writes
      close (file[0]);
      n = write (file[1], &cW, 1);
      printf ("Wrote %d bytes: %c\n", n, cW);
    }
  }
  exit(EXIT_SUCCESS);
}
```

Descrizione del codice (pipe, clse, read e write) ...

4. Dato il seguente grafo di precedenza realizzarlo utilizzando il **minimo** numero possibile di semafori. Tutti i processi devono essere considerati ciclici. Si utilizzino le primitive `init`, `signal`, `wait` e `destroy`. Riportare l'inizializzazione e la distruzione dei semafori e il corpo dei processi $(P_1, \ldots, P_8)$. Indicare inoltre quali e quanti



semafori si dovrebbero utilizzare nel caso in cui i processi fossero **aciclici** e non fossero presenti gli archi di ritorno (quelli orientati verso l'alto). Motivare la soluzione indicata.

Gli archi P_3–P_7 e P_7–P_2 sono superflui.

Prima dell'esecuzione:

```
sem_t s1, s2, ..., s8;
...
init (s1, 1);
init (s2, 0);
...
init (s8, 0);
```

Al termine dell'esecuzione:

```
destroy (s1);
destroy (s2);
...
destroy (s8);
```

P1

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

P2

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

P3

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

P4

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

P5

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

P6

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

P7

```
while (1) {
  wait (s7);
  wait (s7);
  printf ("P7\n");
  signal (s8);
}
```

P8

```
while (1) {
  wait (s8);
  printf ("P8\n");
  signal (s1);
}
```

5. Uno script BASH riceve 4 parametri. I primi tre parametri sono nomi di direttori, `dir1`, `dir2` e `dir3`; il quarto parametro è un numero intero n.

   Il numero di parametri va controllato. Se il direttorio `dir3` non esiste occorre crearlo. Lo script deve quindi rintracciare nei direttori `dir1` e `dir2` tutti i file che hanno lo stesso nome, estensione `''txt''` e più di n righe.

   Lo script deve quindi creare nel direttorio `dir3`:

   - una versione dei file con estensione `eq` in cui vengono memorizzate le righe uguali dei due file originari.
   - una versione dei file con estensione `dif` che memorizza solo le righe diverse dei due file.
   - una versione dei file con estensione `cat` che memorizza la concatenazione dei due file.

   Soluzione 1:

```
#!/bin/bash
if [ $# -ne 4 ]
then
  echo "usage: $0 dir1 dir2 dir3 n"
  exit 1
fi
if [ ! -d $3 ]; then
  mkdir $3
fi
for file in $(ls $1/*.txt); do
    name=$(basename $file ".txt")
    if [ -f "$2/$name.txt" ]; then
        n1=$(cat $file | wc -l)
        n2=$(cat "$2/${name}.txt" | wc -l)
        if [ $n1 -ge $4 -a $n2 -ge $4 ]; then
            max=$n1
            if [ $n1 -lt $n2 ]; then
                max=$n2
            fi
            for((i=1; i<=$max; i++)); do
                if [ $i -le $n1 -a $i -le $n2 ]; then
                    line1=$(sed -n "${i}p" "$file")
                    line2=$(sed -n "${i}p" "$2/${name}.txt")
                    if [ $line1 == $line2 ]; then
                        echo $line1 >> "$3/${name}.eq"
                    else
                        echo $line1 >> "$3/${name}.dif"
                        echo $line2 >> "$3/${name}.dif"
                    fi
                elif [ $i -gt $n1 ]; then
                    line2=$(sed -n "${i}p" "$2/${name}.txt")
                    echo "$line2" >> "$3/${name}.dif"
                else
                    line1=$(sed -n "${i}p" "$file")
                    echo "$line1" >> "$3/${name}.dif"
                fi
            done
            cat $file "$2/${name}.txt" > "$3/${name}.cat"
        fi
    fi
done
```

Soluzione 2:

```
#!/bin/bash
if [ $# -ne 4 ]
then
  echo "usage: $0 dir1 dir2 dir3 n"
  exit 1
fi
if [ ! -d $3 ]; then
  mkdir $3
fi
for file in $(ls $1/*.txt); do
    name=$(basename $file ".txt")
    if [ -f "$2/$name.txt" ]; then
        n1=$(cat $file | wc -l)
        n2=$(cat "$2/${name}.txt" | wc -l)
        if [ $n1 -ge $4 -a $n2 -ge $4 ]; then
            while read line; do
                grep -q -e "^$line$" "$2/$name.txt"
                if [ $? -eq 0 ]; then
                    echo $line >> "$3/${name}.eq"
                else
                    echo $line >> "$3/${name}.dif"
                fi
            done < $file
            while read line; do
                grep -q -e "^$line$" "$3/${name}.eq"
                if [ $? -eq 1 ]; then
                    echo $line > "$3/${name}.dif"
                fi
            done < "$2/$name.txt"
            cat $file "$2/${name}.txt" > "$3/${name}.cat"
        fi
    fi
done
```

Soluzione 3:

```
#!/bin/bash
if [ $# -ne 4 ]
then
  echo "Usage $0 dir1 dir2 dir3 n"
fi
if [ ! -d $3 ]
then
 mkdir $3
fi
for f1 in ‘find $1 -maxdepth 1 -type f -name "*.txt"‘
do
  fout1=‘basename $f1‘
  fout2=‘basename $f1 .txt‘
  if [ -e $2/$fout1 ]
  then
    nlf1=‘cat $f1 | wc -l‘
    nlf2=‘cat $2/$fout1 | wc -l‘
   if [[ "$nlf1" -gt $4 && "$nlf2" -gt "$4" ]]
     then
       cat $f1 $2/$fout1 > $3/$fout2".cat"
       while read line
       do
       grep "$line" $2/$fout1 > .tmp1
         if [ $? -eq 0 ]
         then
            echo $line >> $3/$fout2".eq"
         else
            echo $line >> $3/$fout2".dif"
         fi
       done < $f1
    fi
  fi
done
```

6. Un testo memorizzato in un file ha un contenuto simile a quello del seguente esempio:

```
Nel mezzo del cammin di nostra vi-
ta mi ritrovai per una selva oscu-
ra che' la diritta via era smar-
rita. Ahi quanto a dir qual era e' cosa
dura. Esta selva selvaggia e aspra e forte
che nel pensier rinova la paura!
```

Si scriva uno script AWK in grado di riformattare il testo secondo le seguenti regole:

- la suddivisione in sillabe (evidenziate dalla presenza di un carattere `''-''` al termine di una riga), devono essere eliminate, riunendo la parola sillabata sulla riga successiva.
- A tutti i segni di punteggiatura punto `''.''` deve seguire un `''a capo''`. La parte di riga che segue il punto deve essere memorizzata sulla riga successiva.

Per esempio l'applicazione delle regole descritte sull'esempio riportato fornirebbe il seguente risultato:

```
Nel mezzo del cammin di nostra
vita mi ritrovai per una selva
oscura che' la diritta via era
smarrita.
Ahi quanto a dir qual era e' cosa dura.
Esta selva selvaggia e aspra e forte
che nel pensier rinova la paura!
```

Suggerimento: si ricorda che la funzione `gsub (regExp, str [,src])` sostituisce ogni occorrenza dell'espressione regolare `regExp` con `str` nella stringa `src` (oppure in `$0` se `src` non è presente).

```
BEGIN {
  precword=""
  found=0;
}
{
  i=1
  limit=NF
  if (found==1) {
    sub("-","",precword)
    concat=precword""$1
    if (concat~/.*\./)
      printf concat"\n"
    else
      printf concat" "
    i++
    found=0;
  }
  if ($NF~/.*-/) {
    precword=$NF
    found=1;
    limit--
  }
  while(i<=limit) {
    if ($i~/.*\./||i==limit) {
      printf $i"\n"
    } else {
      printf $i" "
    }
    i++
  }
}
```