OK PDF

Caselle riservate

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

Sistemi Operativi

Compito d'esame

29 Giugno 2016

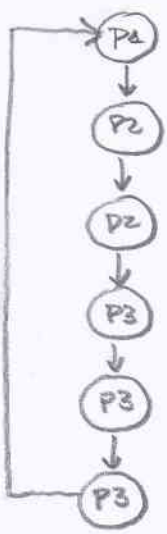
Matricola _____ Cognome _____ Nome _____

Docente: ○ Quer ○ Sterpone

Non si possono consultare testi, appunti o calcolatrici a parte i formulari distribuiti dal docente. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 100 **minuti.**

1. Un programma concorrente è costituito da 3 processi (P_1, P_2, P_3) **ciclici**, **non** ricorsivi, che **non** si richiamano a vicenda e di cui è presente un'unica istanza in esecuzione. I processi sono tali per cui P_1 deve essere eseguito per primo una sola volta, P_2 deve essere eseguito subito dopo P_1 e per due volte, P_3 deve essere eseguito subito dopo P_2 e per tre volte. Al termine dell'esecuzione di P_3 il procedimento deve riprendere dall'inizio con l'esecuzione di P_1. Si rappresenti il grafo di precedenza del sistema e si scriva il programma (in pseudo-codice) illustrandone i meccanismi di sincronizzazione utilizzando il minimo numero di semafori possibile.

   Come è possibile aggiungere al sistema precedente un nuovo processo ciclico e non ricorsivo P_4 che può essere eseguito in concorrenza con P_1 e P_3 ma in **mutua esclusione** con P_2?




```
init(S1,1);
init(S2,0);
init(S3,0);
int m1=0;
int m2=0;
```

AL TERMINE:
```
destroy(S1);
destroy(S2);
destroy(S3);
```

```
P1() {
  while(1) {
    wait(S1);
    PRINT("P1");
    signal(S2);
    signal(S2);
  }
}
```

```
P2() {
  while(1) {
    wait(S2);
    PRINT("P2");
    m1++;
    if(m1==2) {
      m1=0;
      signal(S3);
      signal(S3);
      signal(S3);
    }
  }
}
```

```
P3() {
  while(1) {
    wait(S3);
    PRINT("P3");
    m2++;
    if(m2==3) {
      m2=0;
      signal(S1);
    }
  }
}
```

```
init(ME,1);
P4() {
  while(1) {
    wait(ME);
    PRINT("P4");
    signal(ME);
  }
}
destroy(ME);
```

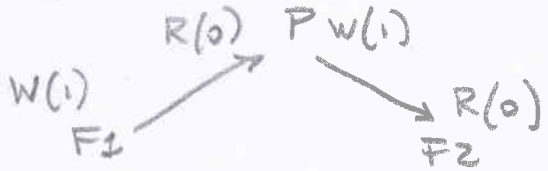
2. Si descrivano le caratteristiche principali delle *pipe* per la comunicazione e la sincronizzazione tra processi.

Se ne illustri l'utilizzo realizzando il seguente programma in linguaggio C. Un processo P genera due figli F_1 e F_2. F_1 legge da tastiera una stringa di lunghezza uguale a 10 caratteri, la trasmette al padre P su una prima pipe e termina. P inverte l'ordine dei caratteri all'interno della stringa (ovvero, ad esempio la stringa "0123456789" diventa "9876543210") e trasmette la stringa su una seconda pipe a F_2. F_2 visualizza la stringa ricevuta su standard output e termina.

IPC = Inter Process Communication
- scambio messaggi
- memoria condivisa

Half Duplex Pipe = Flusso dati tra 2 processi (P1 e P2)

Simile a file con R/W su 2 descrittori interi
Un processo scrive a un estremo (P1)
Un processo legge all'altro estremo (P2)
Half. duplex = i dati fluiscono da P1 a P2 o viceversa (NON contemporaneamente)
P1 e P2 devono avere un parente comune.




```
#include ...
int main() {
    int pipe1[2], pipe2[2];
    if (pipe(pipe1) == (-1)) { ... errore ... }
    if(pipe(pipe2) == (-1)) { ... errore ... }
    pid1 = fork();
    if (pid1 == 0) {
        figlio1(pipe1);
    } else {
        pid2 = fork();
        if (pid2 == 0) {
            figlio2(pipe2);
        } else {
            padre(pipe1, pipe2); wait(); wait();
        }
    }
    return;
}
```

```
void figlio1(int pipe1[2]) {
    char s[11];
    close(pipe1[0]);

    printf("F1:");
    scanf("%s", s);
    write(pipe1[1], s, 11 * sizeof(char));
    exit(EXIT_SUCCESS);
}
```

```
void figlio2(int pipe2[2]) {
    char s[11];
    close(pipe2[1]);
    read(pipe2[0], s, 11 * sizeof(char));
    printf("F2: %s\n", s);
    exit(EXIT_SUCCESS);
}
```

```
void padre(int pipe1[2], int pipe2[2]) {
    char c, s[11];
    int i;
    close(pipe1[1]);
    close(pipe2[0]);
    read(pipe1[0], s, 11 * sizeof(char));
    for(i=0; i<5; i++) {
        c = s[i]; s[i] = s[10-i-1]; s[10-i-1] = c;
    }
    write(pipe2[1], s, 11);
    return;
}
```

3. Si illustri il problema del "*Produttore e Consumatore*" generalizzandolo come segue. Esistono P produttori, due insiemi di consumatori C_1 e C_2 e due code Q_1 e Q_2, di lunghezza uguale a N_1 e N_2, rispettivamente. Le code sono utilizzate dai produttori e dai consumatori per comunicare. I P produttori producono tutti prima un oggetto sulla coda Q_1 e successivamente un oggetto sulla coda Q_2. I consumatori C_1 consumano solo oggetti estratti da Q_1. I consumatori C_2 consumano solo oggetti estratti da Q_2. Si rappresenti un possibile schema di funzionamento in pseudo-codice e si indichi la funzione dei vari semafori motivandone l'utilizzo.

```
init (full1, 0);
init (empty1, N1);
init (MEp1, 1);
init (MEp2, 1);
init (MEC1, 1);
init (MEC2, 1);

init (full2, 0);
init (empty2, N2);
```

```
P() {
    message m;
    while (1) {
        produce (&m);
        wait (empty1);
        wait (MEp1);
        enqueue1 (m);
        signal (MEp1);
        signal (full1);
        produce (&m);
        wait (empty2);
        wait (MEp2);
        enqueue2 (m);
        signal (MEp2);
        signal (full2);
    }
}
```

```
C1() {
    message m;
    while (1) {
        wait (full1);
        wait (MEC1);
        m = dequeue1();
        signal (MEC1);
        signal (empty1);
        consume (m);
    }
}
```

```
C2() {
    message m;
    while (1) {
        wait (full2);
        wait (MEC2);
        m = dequeue2();
        signal (MEC2);
        signal (empty2);
        consume (m);
    }
}
```

4. Scrivere uno script BASH che opera seguendo le specifiche successive. Lo script riceve sulla riga di comando un unico parametro, corrispondente al nome di un file. Tale file memorizza, su righe successive, un numero indefinito di path a file regolari, direttori, link e altri file speciali. Lo script deve visualizzare su standard output il numero di file regolari di proprietà dell'utente che ha eseguito lo script e che abbiano dimensione maggiore di 1 KByte. Se un path si riferisce a un direttorio, lo script deve ripetere la stessa operazione all'interno del direttorio stesso, visualizzando lo stesso risultato accompagnato dal nome del direttorio. Al termine di tutte le operazioni, lo script deve anche visualizzare il numero di file che soddisfano le condizioni indicate e il numero totale di righe memorizzate in tali file.

   Si osservi che occorre gestire solo i direttori del primo livello e non procedere in maniera ricorsiva. Inoltre, il nome dello user può essere ottenuto con i comandi `whoami` oppure `echo $USER`. Infine si ricorda che il formato del comando `ls -l` è il seguente:

```
total 400
-rw-rw-r-- 1 quer quer  60928 Jan 25 17:26 20160126.doc
...
-rw-rw-r-- 1 quer quer   4315 Jun 27 13:49 20160701pgrm.tex
-rw-rw-r-- 1 quer quer    302 Jun 23 11:42 Makefile
```

```
#!/bin/bash

# Check parameters
if [ $# -ne 1 ]
then
      echo "Usage: $0 <input file>"
      exit 1
fi

#Variables initialization
n=0
rows=0

#Process entries
while read path; do

    #File entry
    if [ -f $path ]; then
        owner=$(ls -l $path | cut -d " " -f 3)
        size=$(cat $path | wc -c)
        if [ $owner == $USER ] && [ $size -gt 6 ]; then
            tmp=$(cat $path | wc -l)
            let "n=n+1"
            let "rows=rows+tmp"
        fi

    #Directory entry
    elif [ -d $path ]; then
        echo "$path:"
        ndir=0
        rowsdir=0
        for file in $(ls $path); do
            if [ -f "$path/$file" ]; then
                owner=$(ls -l "$path/$file" | cut -d " " -f 3)
                size=$(cat "$path/$file" | wc -c)
                if [ $owner == $USER ] && [ $size -gt 6 ]; then
                    tmp=$(cat "$path/$file" | wc -l)
                    let "ndir=ndir+1"
                    let "rowsdir=rowsdir+tmp"
                fi
            fi
        done
        echo "  Number of files: $ndir"
        echo "  Number of rows: $rowsdir"
        let "n=n+ndir"
        let "rows=rows+rowsdir"
    fi
done < $1

#Output total
echo "Total:"
echo "  Number of files: $n"
echo "  Number of rows: $rows"
```

5. Un file di tipo "elenco" memorizza, su ciascuna riga, il nome di un file, il suo percorso e la sua dimensione in byte. Un file di tipo "cancella" memorizza l'elenco delle righe del file da cancellare. Il seguente è un esempio di tali file.

File elenco:

```
main.c      /home/alfa/project/src       2134
test.c      /home/alfa/project/src       1365
libutils.h  /home/alfa/project/src/libs  1197
libtime.h   /home/alfa/project/src/libs  4504
run1.log    /home/alfa/project/test      1277
```

File cancella:

```
main.c
run1.log
```

Si scriva uno script AWK in grado di:

- Cancellare dal file di tipo "elenco" le righe indicate nel file "cancella".
- Visualizzare (su standard output) l'elenco delle righe cancellate, e al termine, lo spazio totale occupato dai file le cui righe sono state cancellate.

Nel caso relativo ai file precedenti, al termine dell'esecuzione dello script il file "elenco" sarà il seguente:

```
test.c      /home/alfa/project/src       1365
libutils.h  /home/alfa/project/src/libs  1197
libtime.h   /home/alfa/project/src/libs  4504
```

e lo script avrà visualizzato su standard output:

```
main.c    /home/alfa/project/src   2134
run1.log  /home/alfa/project/test  1277
Totale:  3411 Byte
```

I nomi dei file "elenco" e "cancella" possono essere ricevuti sulla riga di comando oppure impostati in maniera alternativa a scelta.

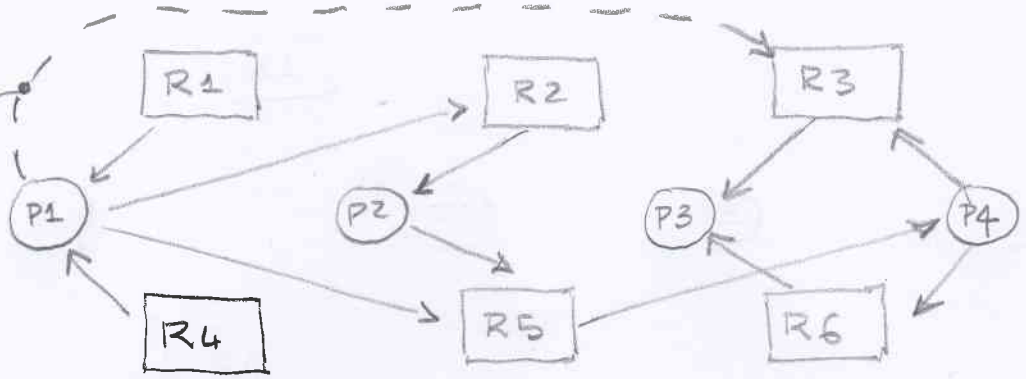
```
#!/usr/bin/awk -f

BEGIN {
    while(getline < ARGV[2]){
        cancella[$1] = $1
    }
    ARGC--
    i=0
    totbytes=0
}

{
    if(!($1 in cancella)){
        out[i++] = $0
    } else {
        printf("%s\n", $0)
        totbytes+=$3
    }
}

END {
    printf("") > ARGV[1]
    for(j=0;j<i; j++){
        printf("%s\n", out[j]) >> ARGV[1]
    }
    printf("Totale: %d Byte\n", totbytes)
}
```

6. In un sistema concorrente il sistema operativo deve gestire 6 risorse ($R_1, R_2, R_3, R_4, R_5, R_6$) con istanze unitarie. In un certo istante sono in esecuzione sul sistema i seguenti quattro processi:

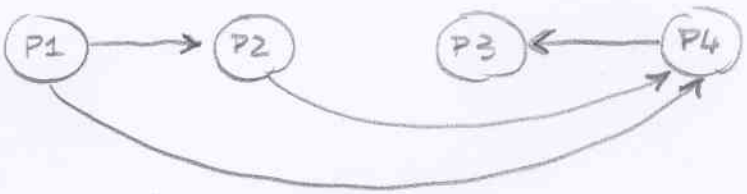
   - Il processo P_1, che ha ottenuto l'assegnazione delle risorsa R_1 e R_4 e ha richiesto le risorse R_2 e R_5.
   - Il processo P_2, che ha ottenuto l'assegnazione della risorsa R_2 e ha richiesto la risorsa R_5.
   - Il processo P_3, che ha ottenuto l'assegnazione delle risorse R_3 e R_6.
   - Il processo P_4, che ha ottenuto l'assegnazione della risorsa R_5 e ha richiesto le risorse R_3 e R_6.

   In futuro il processo P_1 richiederà anche una istanza di R_3. Si rappresentino il grafo di allocazione delle risorse, il grafo di attesa e il grafo di rivendicazione. In generale, si indichi come è possibile rilevare e ripristinare una situazione di deadlock utilizzando il grafo di allocazione delle risorse. Nello specifico, si indichi se il sistema indicato è in stallo oppure perchè non lo è.

Grafo di Rivendicazione



Grafo di Attesa



Rilevazione: verifica presenza cicli grafo di allocazione
NO cicli → NO stallo
cicli → deadlock SE c'è solo 1 istanza x risorse
altrimenti non si può dire

Nel nostro caso ∄ cicli: finisce P3; P4 acquisisce R3 e R6 poi finisce; P2 acquisisce R5 poi finisce; P1 acquisisce R2 e R5 poi finisce.

Ripristino: Possibilità diverse
- Terminare tutti i processi in stallo
- Terminare 1 processo alla volta tra quelli in stallo
- Prelazionare le risorse a quelli in stallo

...