Esercizi 2

Esercizio 1

Quella che segue è una corretta implementazione della mutua esclusione?

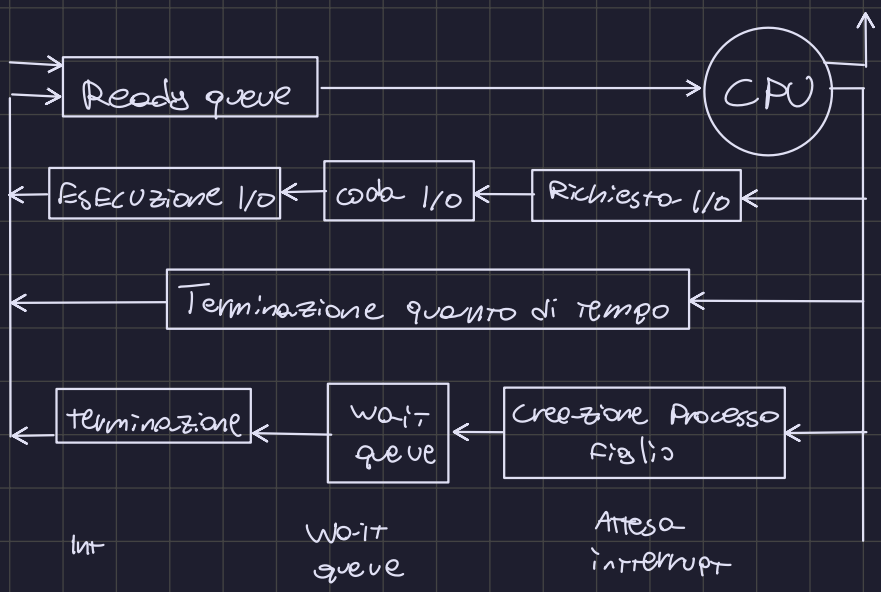
```
boolean test_and_set(boolean *target) {
    boolean old = *target;
    *target = true;
    return old;
}


boolean lock= false; // lock è settato a false per tutti i processi


// Codice per ciascun processo P[i]

P[i]{
    while (true) {
        // Sezione non critica
        // Attesa per entrare nella sezione critica
        while (test_and_set(&lock)==lock) ;
        // Sezione critica
        lock = false;
    }
}
```

Non è corretto perchè nel while viene prima assegnato a lock il suo valore e poi viene confrontato con il return della test and set, quindi si avrebbe false == false e si entra in un ciclo infinito.

Esercizio 3

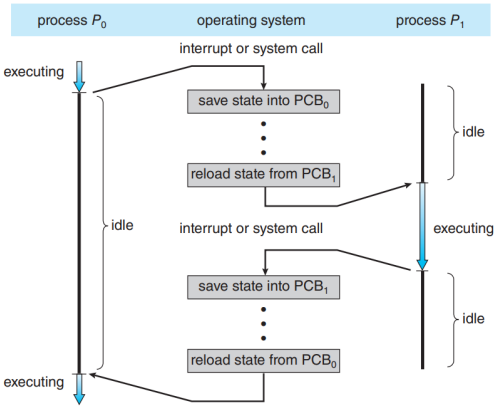
Illustrare graficamente e a parole il diagramma di accodamento per lo scheduling dei processi.

(Soluzione: sezione 3.2.1 e figura 3.5 del libro di testo)



Esercizio 4

Illustrare sinteticamente il seguente diagramma (soluzione sezione 3.2.3 libro di testo)



Questo rappresenta il cambio di contesto. Quando un processo viene interrotto viene salvato il PCB del processo in memoria e viene ripristinato il PCB di un altro processo salvato per metterlo in esecuzione.

Esercizio 7

```c
#include<stdio.h>
#include<signal.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
void f(int);
int main (void){
    signal(SIGCHLD,f);
    int pp;
    if( (pp = fork()) == 0){
        puts("a");
        fflush(stdout);
        while(1);
    }
    else {
        kill(pp,SIGTERM);
        while(1);
    }
}
void f(int x){
    printf("%d\n",x);
    while(wait (NULL)  != -1);
    puts("pippo\n");
    exit(50);
}
```

Si consideri il seguente programma

1. viene stampato **a** ? SI
2. Il programma termina? SI
3. Il processo padre termina? SI
4. Il processo figlio termina? SI
5. Viene generato uno zombie sino a che il padre è in esecuzione? NO
6. Viene stampato **pippo**? CERTO
7. Il processo padre esegue un ciclo di wait per aspettare la morte del figlio? SI, ma nella funzione F
8. PROBABILE

Esercizio 8

```c
#include <stdio.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#define WRITE 1
#define READ 0
int main (void){
    int pid,fid,  st;
    int fd[2];
    pipe(fd);
    if((pid = fork()) == 0){
        int i;
        close(fd[READ]);
        for(i=1;i<10;i++){
            //sleep(1);
            write(fd[WRITE], (char *)&i, sizeof(int));
            sleep(1);
        }
        printf("pippo \n");
        exit(40);
    }
    else{
        if((pid = fork()) == 0){
            int i;
            close(fd[READ]);
            for(i=-1;i> -10;i--){
                write(fd[WRITE], (char *)&i, sizeof(int));
                sleep(2);
            }
            printf("pluto \n");
            exit(40);
        }
        else{
            if((pid = fork()) == 0) {
                int j, k =0;
                int v[18];
                int letti=1;;
                close(fd[WRITE]);
                while ((letti = read(fd[READ], (char *)&j, sizeof(int))) != 0){
                    v[k]=j;k++;
                }
                for(j=0;j<k;j++){
                    printf("v[%d]=%d; ",j,v[j]);fflush(stdout);}
                exit(50);
            }
            else{ close(fd[READ]);close(fd[WRITE]);
                while(wait(&st) != -1);
                printf("\n... i figli sono terminati");
            }
        }
    }
}
```

Spiegare il funzionamento del seguente programma

Il programma genera 3 figli. I primi 2 sono i produttori che generano dei numeri e li scrivono nel file descriptor della pipe aperta. Il terzo figlio legge dalla read della pipe tutti i numeri e li stampa sul terminale. Quando i primi 2 figli hanno finito stampano rispettivamente "pippo" e "pluto". Il padre aspetta che i figli terminino con una wait in while.

Esercizio 9

Tre processi `P1`, `P2` e `P3` devono eseguire un lavoro in una sezione critica ed essere sincronizzati in modo che:

1. `P1` esegua una parte del lavoro.
2. `P2` esegua il suo lavoro **solo dopo** che `P1` ha terminato.
3. `P3` esegua il suo lavoro **solo dopo** che sia `P1` che `P2` hanno completato le loro operazioni.
4. L'accesso alls sezione critica condivisa deve essere garantito con mutua esclusione.

Dire se la seguente è una soluzione al problema mostrando che vengono soddisfatti i vincoli da 1 a 4.

```
semaphore S1 = 0    // Sincronizzazione tra P1 e P2
semaphore S2 = 0    // Sincronizzazione tra P2 e P3
semaphore mutex = 1 // Per garantire mutua esclusione
P1:
    wait(mutex)
    print("P1 inizia lavoro")
    esegui_operazione_P1()
    print("P1 termina lavoro")
    signal(mutex)
    signal(S1)
P2:
    wait(S1)
    wait(mutex)
    print("P2 inizia lavoro")
    esegui_operazione_P2()
    print("P2 termina lavoro")
    signal(mutex)
    signal(S2)
P3:
    wait(S2)
    wait(mutex)
    print("P3 inizia lavoro")
    esegui_operazione_P3()
    print("P3 termina lavoro")
    signal(mutex)
```

È corretto