Esercizi

Esercizio 1

Considera i seguenti processi che arrivano nello stesso istante (tempo 0). Le informazioni sui processi sono riportate nella tabella seguente:
Disegna il diagramma di Gantt per ciascuno dei seguenti algoritmi di scheduling:

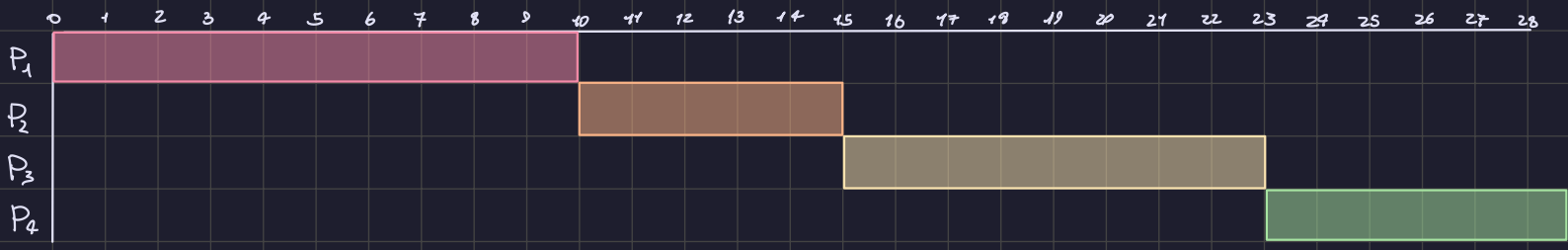
- First-Come, First-Served (FCFS)
- Shortest Job Next (SJN)
- Round Robin (RR) con un quantum di 4 ms
- Scheduling con priorità (Priorità decrescente, in caso di parità usa FCFS come criterio secondario).

Per ogni algoritmo, calcola i seguenti parametri:

- Tempo di completamento (Turnaround Time) per ogni processo.
- Tempo di attesa (Waiting Time) per ogni processo.
- Tempo di risposta (Response Time) per ogni processo.

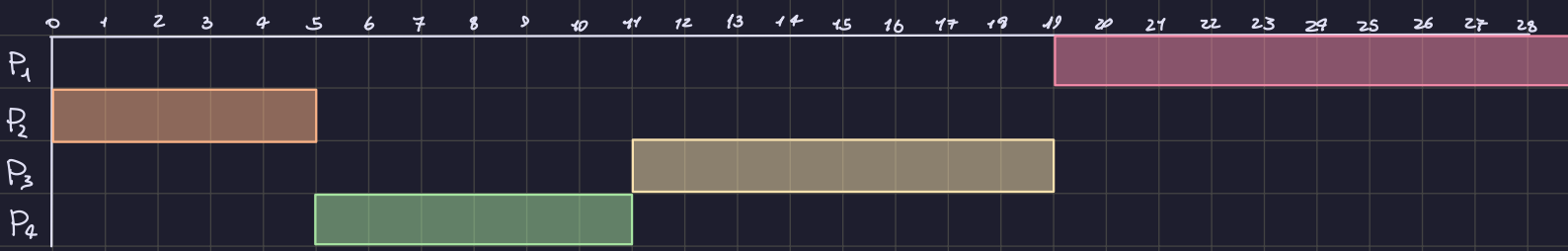
Processo	Tempo di Burst (ms)	Priorità (1=massima)
P1	10	2
P2	5	1
P3	8	3
P4	6	2

FCFS



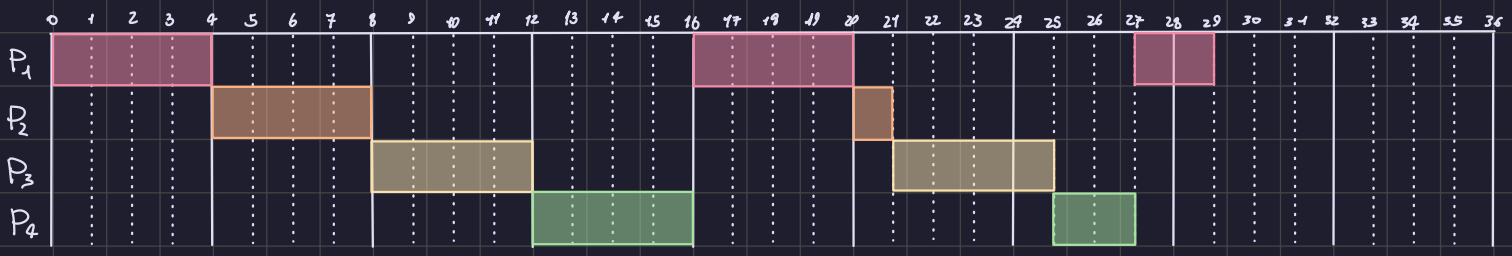
	T_r	T_w	T_T
P_1	0	0	10
P_2	10	10	15
P_3	15	15	23
P_4	23	23	29

SJF



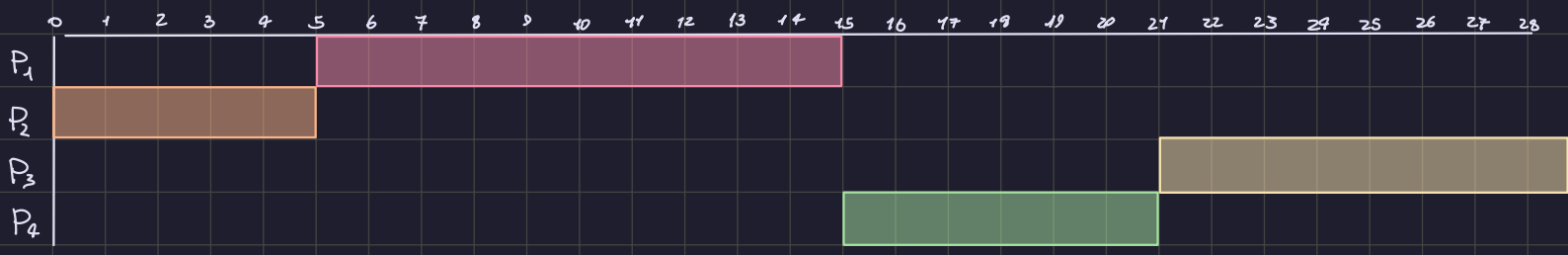
	T_r	T_w	T_T
P_1	19	19	29
P_2	0	0	5
P_3	11	11	19
P_4	5	5	11

Round Robin $q = 4ms$



	T_r	T_w	T_T
P_1	0	19	29
P_2	4	16	21
P_3	8	17	25
P_4	12	21	27

FCFS



	T_r	T_w	T_T
P_1	5	5	15
P_2	0	0	5
P_3	21	21	29
P_4	15	15	21

Esercizio 2

Implementare la mutua esclusione con le mailbox

```
void main() {
  create mailbox (box); // Creo una mailbox chiamata box
  send(box, NULL); // Inizializzazione
}

void exec(message msg) {
  recieve(box, msg); // Se non ci sono i messaggi il programma viene bloccato
  // Sezione critica in cui si entra solo se il processo ha un messaggio ricevuto
  send(box, msg); // Sblocca il prossimo processo inviando un messaggio
  // Sezione non critica
}
```

Esercizio 3

Consider the following processes P1 and P2 that update the value of the shared variables, x and y, as follows:

```
Process P1 :
(  performs the operations:
       x := x * y
       y ++
)
LOAD R1, X
LOAD R2, Y
MUL R1, R2
STORE X, R1
INC R2
STORE Y, R2
```

```
Process P2 :
(  performs the operations:
       x ++
       y := x * y
)
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3
STORE X, R3
STORE Y, R4
```

Assume that the initial values of x and y are 2 and 3 respectively. P1 enters the system first and so it is required that the output is equivalent to a serial execution of P1 followed by P2. The scheduler in the uniprocessor system implements a pseudo-parallel execution of these two concurrent processes by interleaving their instructions without restricting the order of the interleaving.

a. If the processes P1 and P2 had executed serially, what would the values of x and y have been after the execution of both processes?

b. Write an interleaved concurrent schedule that gives the same output as a serial schedule.

c. Write an interleaved concurrent schedule that gives an output that is different from that of a serial schedule.

a) Se i processi vengono eseguiti uno dopo l'altro, il valore di x = 7, y = 28
b) P1,P2,P1,P2 (del codice in c)
c) P1,P2,P2,P1 (cel codice in c)

Esercizio 4

Considerate la seguente **nuova** formulazione

```
void newWait(*s){
    if (s->value > 0){
        s->value --;
    }
    else {
        block();
        place this process in s->list;
    }
}

void newSignal (*s){
    if (there is at least one process blocked on semaphore *s) {
        remove a process P from s->list;
        wakeup(P)
     }
    else s->value ++;
}
```

```
wait(semaphore *S) {
   S->value--;
   if (S->value < 0) {
      add this process to S->list;
      block();
   }
}

signal(semaphore *S) {
   S->value++;
   if (S->value <= 0) {
      remove a process P from S->list;
      wakeup(P);
   }
}
```

Questa nuova formulazione di wait e signal è corretta?

Valori del contatore: Nella definizione proposta, `s.count` rappresenta direttamente il numero di risorse disponibili.
Nella definizione standard, un valore negativo del contatore rappresenta il numero di processi bloccati in attesa della risorsa.

La wait non è corretta perchè il valore del semaforo viene decrementato soltanto se è positivo, in un semaforo il valore diventa anche negativo per contare il numero di processi nella lista d'attesa.

La signal è sbagliata perchè il valore viene incrementato soltanto se c'è un processo bloccato nella lista del semaforo e non si controlla se

Esercizio 5 (tratto dal libro)

A producer process produces information that is consumed by a consumer process. The information is shared by a pool consists of n buffers, each capable of holding one item.

The producer and consumer processes share the following data structures:

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0
```

We assume that the pool consists of n buffers, each capable of holding one item. The **mutex** binary semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.
The empty and full semaphores count the number of empty and full buffers. The semaphore empty is initialized to the value n; the semaphore full is initialized to the value 0.

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0;
pool buffers[N] = 0;

producer() {
  while (1) {
    // Produce item
    wait(empty);
    wait(mutex);
    // Insert item in buffer
    signal(mutex);
    signal(full);
  }
}
```

```
consumer() {
  while (1) {
    wait(full);
    wait(mutex);
    // Remove item from buffer
    signal(mutex);
    signal(empty);
    // Consume item
  }
}
```

Esercizio 7

Spiega perché gli interrupt non sono appropriati per l'implementazione delle primitive di sincronizzazione nei sistemi multiprocessore.

Non sono appropriati perchè se ci sono istruzioni non atomiche esse possono essere interrotte a metà esecuzione e possono creare un comportamento indefinito in base all'ordine in cui vengono schedulati i processi.

Esercizio 8

Perché se le operazioni di semaforo **`wait()`** e **`signal()`** non vengono eseguite in modo atomico, allora la mutua esclusione può essere violata.

Perchè un'istruzione può corrispondere a n istruzioni di assembly, di conseguenza il programma mentre sta eseguendo le funzioni wait o signal può essere interrotto per schedulare l'esecuzione di un altro processo che potrebbe entrare nella sezione critica e questo viola la mutua esclusione.

Esercizio 9

Un ufficio postale ha un numero limitato di sportelli (ad esempio 3) per servire i clienti. Ogni cliente, una volta arrivato, deve attendere che uno degli sportelli sia libero. Quando uno sportello è libero, il cliente lo utilizza e viene servito. Dopo essere stato servito, il cliente lascia lo sportello libero per il cliente successivo.

Implementa una soluzione a questo problema utilizzando sia semafori che le mailbox, tenendo conto dei seguenti vincoli:

1. Non più di 3 clienti possono essere serviti contemporaneamente.
2. I clienti devono aspettare in coda se tutti gli sportelli sono occupati.
3. Ogni cliente deve accedere in modo esclusivo a uno sportello libero.

Semafori:

```
void main() {
  semaphore posti = 3;
  CreaClienti();
}

cliente[i] {
  wait(posti);
  // Usa sportello
  signal(posti);
}
```

Mailbox

```
void main() {
  mailbox create(box, 3);
  CreaClienti();
}

cliente[i] {
  receive(box, NULL);
  // Usa sportello
  send(box, NULL);
}
```

Esercizio 10

Supponiamo di avere tre processi $Proc_0$, $Proc_1$, $Proc_2$
Una risorsa condivisa R e tre semafori `S[0]`, `S[1]`, `S[2]` inizializzati a `1` per accedere alla risorsa.

$Proc_0$ necessita il via libera di `S[2]` e `S[0]` per accedere alla risorsa

$Proc_1$ necessita il via libera di `S[0]` e `S[1]` per accedere alla risorsa

$Proc_2$ necessita il via libera di `S[1]` e `S[2]` per accedere alla risorsa

$Proc_i$

```
while (true){
    wait (S[i] );
    wait (S[(i+1)%3] );
    /* accedi alla risorsa */
    signal (S[i] );
    signal (S[(i+1)%3] );
    /* fa altre cose */
}
```

- Esistono computazioni che portano il sistema in deadlock?

Si, se lo scheduler esegue la prima riga di ogni processo, viene fatta la wait di tutti i semafori e quindi il sistema va in deadlock perchè l'istruzione successiva è un'altra wait che aspetta la liberazione di un semaforo che non avverrà mai.

Esercizio 11

Considera una variante dell'algoritmo di scheduling Round-Robin (RR) in cui gli elementi nella coda dei processi pronti sono puntatori ai blocchi di controllo dei processi (PCB).

a. Quale sarebbe l'effetto di inserire due puntatori allo stesso processo nella coda dei processi pronti?
b. Quali sarebbero i principali vantaggi e svantaggi di questo schema?
c. Come modificheresti l'algoritmo Round-Robin di base per ottenere lo stesso effetto senza duplicare i puntatori?

a) Quel processo viene eseguito più volte in un solo ciclo della lista, come se avesse più priorità.
b) Vantaggi:
   - il vantaggio è che si può aumentare la priorità di un certo processo inserendolo più volte nella coda
   Svantaggi:
   - ci sono puntatori duplicati per aumentare la priorità di un processo
c) Aggiungerei una struttura che contiene il campo priorità e la coda verrà ordinata in base alla priorità di un certo processo.

Esercizio 12

Spiega le differenze nel grado con cui i seguenti algoritmi di scheduling favoriscono i processi brevi:
a. **FCFS**
b. **RR**

a. FCFS: Questo algoritmo dà precedenza ai processi che arrivano per primi, ma potrebbe arrivare prima un processo molto lungo e quindi il tempo di attesa medio è più alto.
b. RR: Questo algoritmo imposta un quanto di tempo che se viene raggiunto interrompe il processo corrente per poter mettere in esecuzione il successivo nella lista. Questo previene che un processo lungo trattenga la CPU per troppo tempo, mentre invece se un processo è più corto del quanto di tempo non subisce prelazione.

Esercizio 13

Qual è il significato del termine *busy waiting (attesa attiva)*? Quali altri tipi di attesa esistono in un sistema operativo? È possibile evitare del tutto il *busy waiting*? Spiega la tua risposta.

Il busy waiting avviene quando un processo non effettua alcun tipo di lavoro rimanendo in esecuzione.
Un altro tipo di attesa è quella passiva, in cui il processo viene spostato in attesa.
Il busy waiting si può evitare se si mette in pausa un processo per poter eseguire altri processi mentre il primo aspetta.

Scrivi un programma in C che:

1. Crea un processo figlio utilizzando la funzione **`fork()`**.
2. Il processo figlio entra in un ciclo infinito stampando un messaggio ogni 2 secondi.
3. Il processo padre attende 10 secondi, quindi invia al processo figlio un segnale di terminazione utilizzando **`kill()`**.
4. Il processo padre attende la terminazione del figlio usando **`wait()`**.
5. Una volta terminato il processo figlio, il padre stampa un messaggio di conferma e termina l'esecuzione.
6.

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
  int pid = fork();
  if (pid < 0) {
    printf("ERORE ERORE !! ATENSIONE\n");
    perror("Motivo");
    exit(-1);
  }

  if (pid == 0) { // Se figlio
    while (1) {
      printf("CACCA\n");
      sleep(2);
    }
  } else {
    sleep(10);
    kill(pid, SIGTERM);
    if (wait(NULL) != -1) {
      printf("MUORI GODOO!\n");
    };
    exit(0);
  }
}
```

Esercizio 15 (non ovvio, ma visto a lezione)
Perché la seguente variante dell'algoritmo di Peterson è errata?

Variabili condivise:

- flag[2]: array di due booleani, inizialmente false. Ogni elemento indica se un processo vuole entrare nella sezione critica.
- turn: variabile intera che indica quale processo ha la precedenza, inizialmente 0.
-

Codice del processo Pi (per i = 0 o 1):

P0

```
while (true){
    turn = 1;
    flag[0] = true;
    while (flag[1] && turn == 1);

          /* critical section */

    flag[0] = false;

   /* remainder section */

}
```

P1

```
while (true){
    turn = 0;
    flag[1] = true;
    while (flag[0] && turn == 0);

          /* critical section */

    flag[1] = false;

   /* remainder section */

}
```

Se si eseguono le istruzioni nell'ordine P_0, P_1, P_1, P_1, P_0, P_0, P_0 si ha che entrambi i processi si trovano nella sezione critica e quindi non è garantita la mutua esclusione.

Esercizio 16

Quali dei seguenti componenti dello stato di un programma sono condivisi tra i thread in un processo multithread?

a. Valori dei registri
b. Memoria heap
c. Variabili globali
d. Memoria dello stack

Esercizio 17

E' possibile combinare lo scheduling con prelazione con quello round robin?

È possibile perchè allo scadere del quanto di tempo il processo che è in esecuzione deve essere interrotto per mettere in esecuzione il successivo nella lista.

Esercizio 18

Si consideri la seguente implementazione dei semafori senza attesa attiva.

E' corretta?

```
typedef struct {
int value;
struct process *list;
} semaphore;

wait(semaphore *S) {
   S->value--;
   while (S->value < 0);
   add this process to S->list;
   sleep();
   }
}

signal(semaphore *S) {
   S->value++;
   if (S->value <= 0) {
      remove a process P from S->list;
      wakeup(P);
   }
}
```

È sbagliato perchè nella wait c'è un while che introduce attesa attiva