REPORT ESERCIZIO S3/L1

TRACCIA: L'esercizio di oggi verte sui meccanismi di pianificazione dell'utilizzo della CPU (o processore).
In ottica di ottimizzazione della gestione dei processi, abbiamo visto come lo scheduler si sia evoluto nel tempo per passare da approccio mono-tasking ad approcci multi-tasking.
Si considerino 4 processi, che chiameremo P1,P2,P3,P4, con i tempi di esecuzione e di attesa input/output dati in tabella. I processi arrivano alle CPU in ordine P1,P2,P3,P4. Individuare il modo più efficace per la gestione e l'esecuzione dei processi, tra i metodi visti nella lezione teorica. Abbozzare un diagramma che abbia sulle ascisse il tempo passato da un instante «0» e sulle ordinate il nome del Processo.

Processo	Tempo di esecuzione	Tempo di attesa	Tempo di esecuzione dopo attesa
P1	3 secondi	2 secondi	1 secondo
P2	2 secondi	1 secondo	--
P3	1 secondo	--	--
P4	4 secondi	1 secondo	--

Per affrontare il problema proposto, è importante analizzare i vari metodi di gestione dei processi e determinare quale sia il più adatto per ottimizzare l'uso della CPU, considerando i tempi di esecuzione e di attesa per i vari processi.

Mono-tasking:

In un sistema mono-tasking, la CPU esegue un solo processo alla volta. Il tempo di esecuzione di ciascun processo viene completato prima di passare al successivo.
Non è particolarmente efficiente se ci sono processi che devono attendere o richiedono risorse in modo asincrono (ad esempio, input/output).

Nel metodo mono-tasking, i processi verranno eseguiti uno dopo l'altro senza preemption. L'ordine di esecuzione sarà il seguente:

- **P1** inizia alle 0 secondi e finisce alle 3 secondi.
- **P2** inizia alle 3 secondi e finisce alle 5 secondi.
- **P3** inizia alle 5 secondi e finisce alle 6 secondi.
- **P4** inizia alle 6 secondi e finisce alle 10 secondi.

Quindi, l'ordine di esecuzione è il seguente:

- **P1** (0-3 secondi)
- **P2** (3-5 secondi)
- **P3** (5-6 secondi)
- **P4** (6-10 secondi)

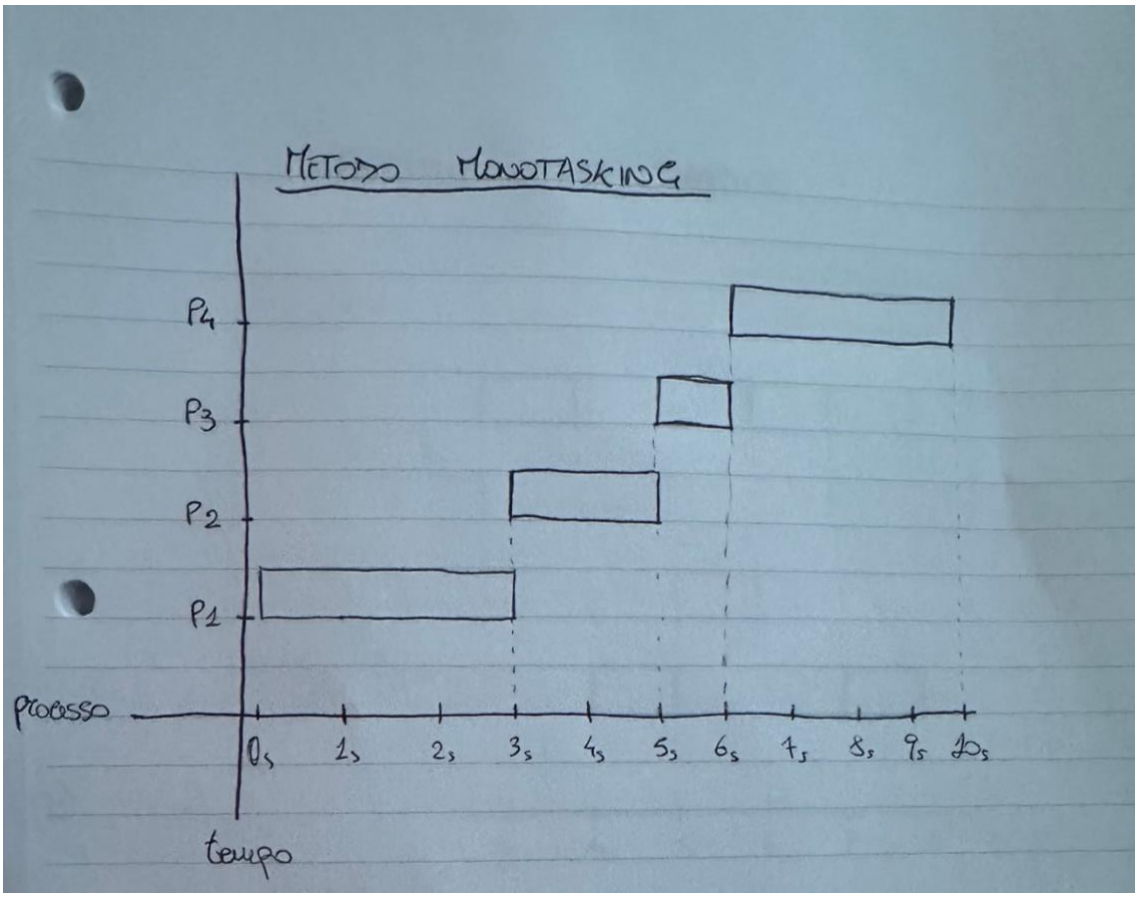
In questo caso, l'ordine è determinato dalla sequenza in cui i processi arrivano alla CPU, ma il tempo totale di attesa (soprattutto per P1) può essere elevato. Non si ottimizza l'utilizzo della CPU quando ci sono attese di input/output.

Tempi di attesa:

- P1: **0 secondi** (inizia subito).
- P2: **3 secondi** (attende che P1 finisca).
- P3: **5 secondi** (attende che P1 e P2 finiscano).
- P4: **6 secondi** (attende che P1, P2 e P3 finiscano).

Somma dei tempi di attesa:
0+3+5+6=14 secondi
Media dei tempi di attesa:
14/4 = 3.5 secondi

Quindi, la **media dei tempi di attesa** per il **mono-tasking** è **3.5 secondi**.
Tempo totale di esecuzione:
L'ultimo processo a finire è P4, che termina a **10 secondi**.
Quindi, il **tempo totale di esecuzione** per il **mono-tasking** è **10 secondi**.



2. Metodo multi-tasking
Nel metodo multi-tasking, possiamo sospendere l'esecuzione di un processo quando è in attesa di I/O, per passare ad un altro processo che è pronto per l'esecuzione. In questo caso, i processi vengono eseguiti in modo più dinamico.
Dati i tempi di attesa di I/O, possiamo fare in modo che il sistema passi da un processo all'altro quando uno di essi è in attesa:

1. **P1** (inizia)

2. Durante l'attesa di P1, **P2** può essere eseguito
3. Quando P1 finisce di attendere, riprende l'esecuzione per 2 secondi.
4. **P3** viene eseguito in seguito (1 secondo).
5. **P4** inizia e prosegue.

Un possibile ordine di esecuzione:

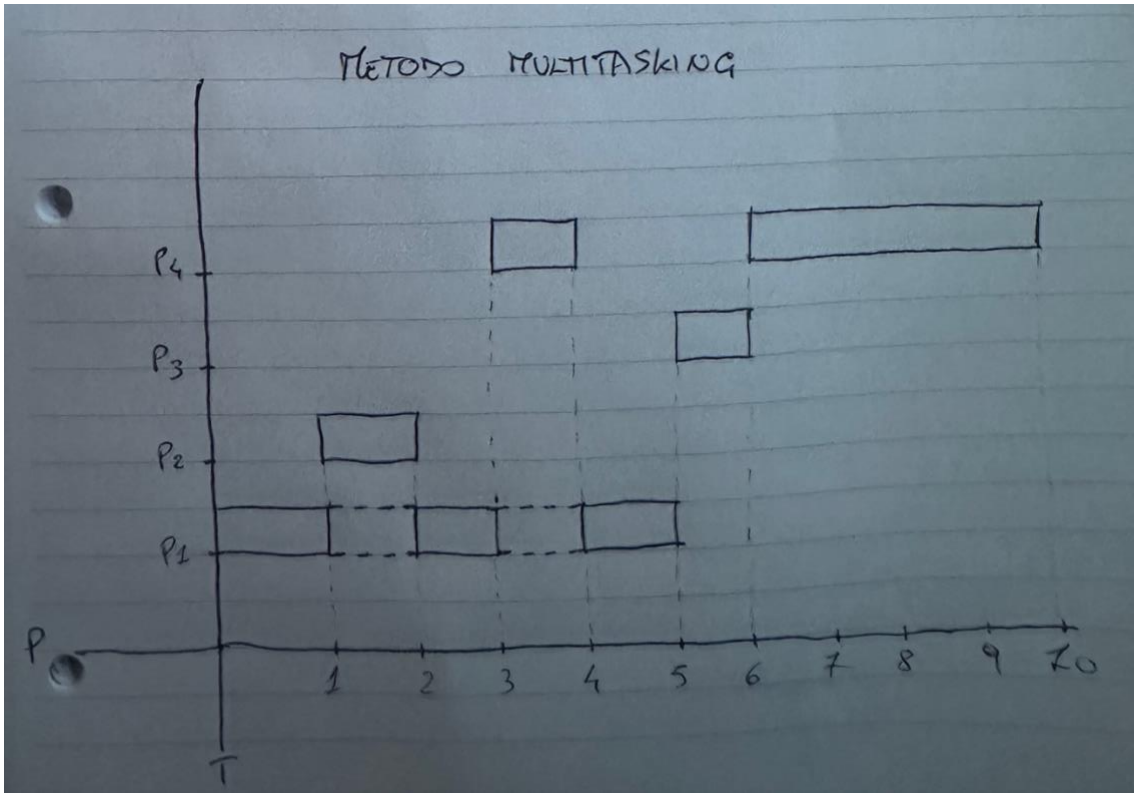
- **P1** (0-1 secondi)
- **P2** (1-2 secondi)
- **P1** (2-3 secondi)
- **P4** (3-4 secondi)
- **P1** (4-5 secondi)
- **P3** (5-6 secondi)
- **P4** (6-10 secondi)

Tempi di attesa:

- P1: **2 secondi** (P1 è in attesa di I/O per 2 secondi).
- P2: **0 secondi** (P2 inizia subito dopo P1, senza attese).
- P3: **5 secondi** (P3 deve attendere che P2 finisca).
- P4: **3 secondi** (P4 deve attendere che P1 e P3 finiscano).

Somma dei tempi di attesa:
2+0+5+3=10 secondi
Media dei tempi di attesa:
10/4 = 2.5 secondi
Quindi, la **media dei tempi di attesa** per il **multi-tasking** è **2.5 secondi**.

Tempo totale di esecuzione:
L'ultimo processo a finire è P4, che termina a **9 secondi**.
Quindi, il **tempo totale di esecuzione** per il **multi-tasking** è **9 secondi**.



Nel caso del time-sharing, la CPU alterna l'esecuzione dei vari processi in intervalli di tempo fissi (quantum). Questo approccio cerca di equilibrare l'uso della CPU tra i vari processi e minimizzare il tempo di attesa.
Supponiamo che ogni quantum di esecuzione sia di 1 secondo.
L'esecuzione dei processi potrebbe essere organizzata come segue:

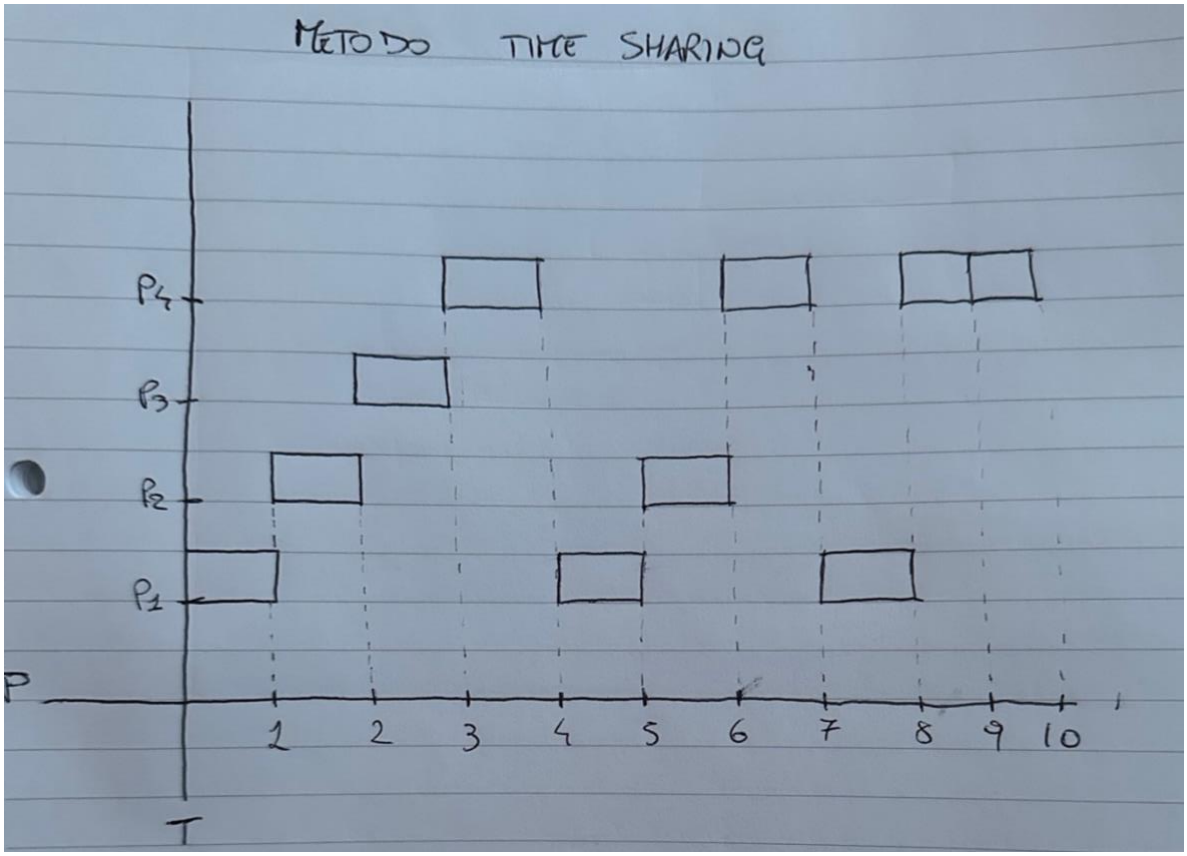
1. **P1** (0-1 secondo)
2. **P2** (1-2 secondi)
3. **P3** (2-3 secondi)
4. **P4** (3-4 secondi)
5. **P1** (4-5 secondi)
6. **P2** (5-6 secondi)
7. **P4** (6-7 secondi)
8. **P1** (7-8 secondi)
9. **P4** (8-9 secondi)
10. **P4** (9-10 secondi)

Nel metodo **time-sharing**, ogni processo riceve un "quantum" di tempo per essere eseguito, e i processi vengono alternati in modo equo.

Tempi di attesa:

- P1: **1 secondo** (P1 attende dopo il suo primo quantum, tra il primo e il secondo turno di esecuzione).
- P2: **0 secondi** (P2 inizia subito dopo P1, senza attese).
- P3: **5 secondi** (P3 deve attendere che P1, P2 e P4 siano eseguiti prima di poter partire).
- P4: **3 secondi** (P4 deve attendere che P1 e P3 finiscano prima di poter riprendere).

Somma dei tempi di attesa:
1+0+5+3=9 secondi
Media dei tempi di attesa:
9/4 = 2.25 secondi
Quindi, la **media dei tempi di attesa** per il **time-sharing** è **2.25 secondi**.
Tempo totale di esecuzione:
L'ultimo processo a finire è P4, che termina a **10 secondi**.
Quindi, il **tempo totale di esecuzione** per il **time-sharing** è **10 secondi**.



Conclusioni:

- **Mono-tasking**: la media dei tempi di attesa è **3.5 secondi**.
- **Multi-tasking:** la media dei tempi di attesa è **2.5 secondi**.
- **Time-sharing**: la media dei tempi di attesa è **2.25 secondi**.

Il metodo **time-sharing** risulta essere il più efficiente per ridurre i tempi di attesa complessivi. Il **multi-tasking** è più efficiente del **mono-tasking**, ma comunque meno ottimizzato rispetto al **time-sharing**.