S3/L1

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. Traccia: 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 secondi	-	-
P4	4 secondi	1 secondo	-

Informazioni fornite

Ci sono 4 processi, ciascuno con tempi di esecuzione, attesa input/output e eventuale esecuzione successiva. I dettagli sono riportati nella tabella:

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	-

Analizziamo i tre casi richiesti con i metodi visti nelle slide: **mono-tasking**, **multi-tasking**, e **time-sharing**, e costruiamo i relativi diagrammi temporali.

1. Mono-tasking

In questo caso, i processi vengono eseguiti uno alla volta, nell'ordine di arrivo (**FIFO – first in first out**). Non c'è preemption (pre rilascio, un processo viene interrotto e portato fuori dalla cpu) e ogni processo viene completato prima che inizi il successivo.

Sequenza di esecuzione:

- **P1**: 0–3 secondi (eseguito interamente).
- **P2**: 3–5 secondi.
- **P3**: 5–6 secondi.
- **P4**: 6–10 secondi.

2. Multi-tasking

In un sistema multi-tasking, ogni processo può subire interruzioni per consentire a un altro processo di essere eseguito. Uno scenario tipico potrebbe essere utilizzare il **Round Robin (RR)** con una **time slice** di 1 secondo. I processi eseguono in ordine ciclico fino al completamento.

Sequenza di esecuzione con RR (time slice = 1 secondo):

- 0–1: **P1**.
- 1–2: **P2**.
- 2–3: **P3**.
- 3–4: **P4**.
- 4–5: **P1** (continua per 2 secondi).
- 5–6: **P4** (continua per 2 secondi).
- 6–7: **P1** (ultima parte, 1 secondo).
- 7–8: **P4** (ultima parte, 1 secondo).

3. Time-sharing

In un sistema di time-sharing, i processi condividono equamente la CPU con brevi time slices, enfatizzando l'interattività e la risposta rapida. Assumiamo un time slice di 1 secondo, simile al multi-tasking con Round Robin (RR).

Sequenza di esecuzione:

- 0–1 secondi: P1
- 1–2 secondi: P2
- 2–3 secondi: P3

- 3–4 secondi: P4
- 4–5 secondi: P1 (continua per 2 secondi)
- 5–6 secondi: P4 (continua per 2 secondi)
- 6–7 secondi: P1 (ultima parte, 1 secondo)
- 7–8 secondi: P4 (ultima parte, 1 secondo)

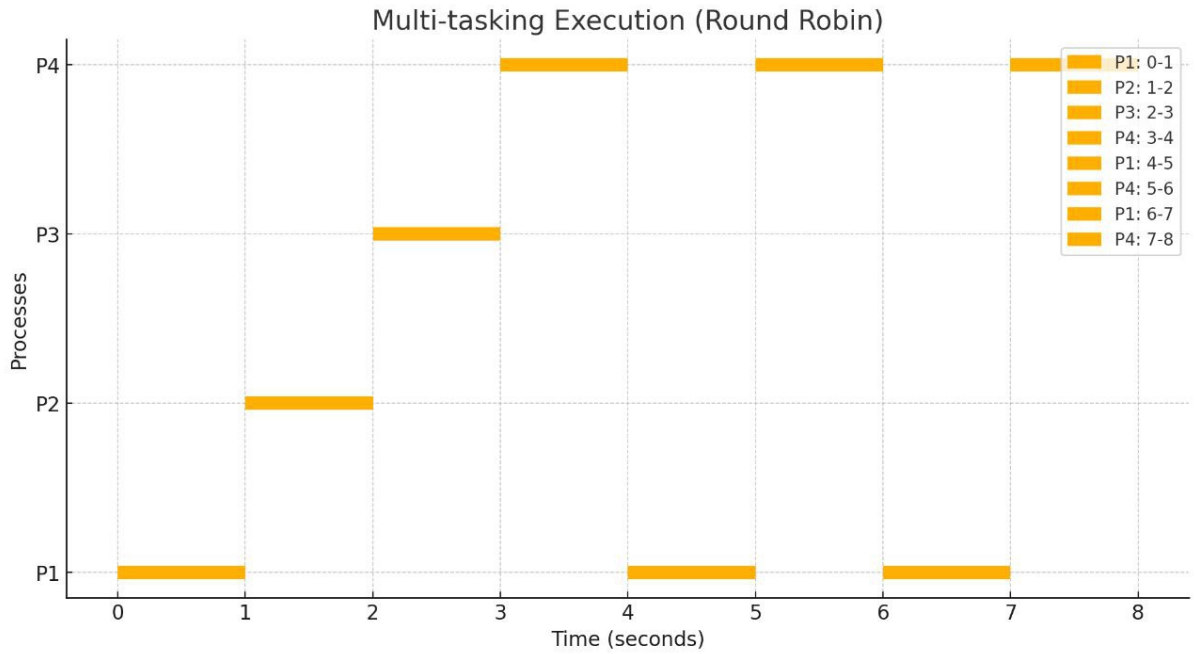
Diagramma temporale:

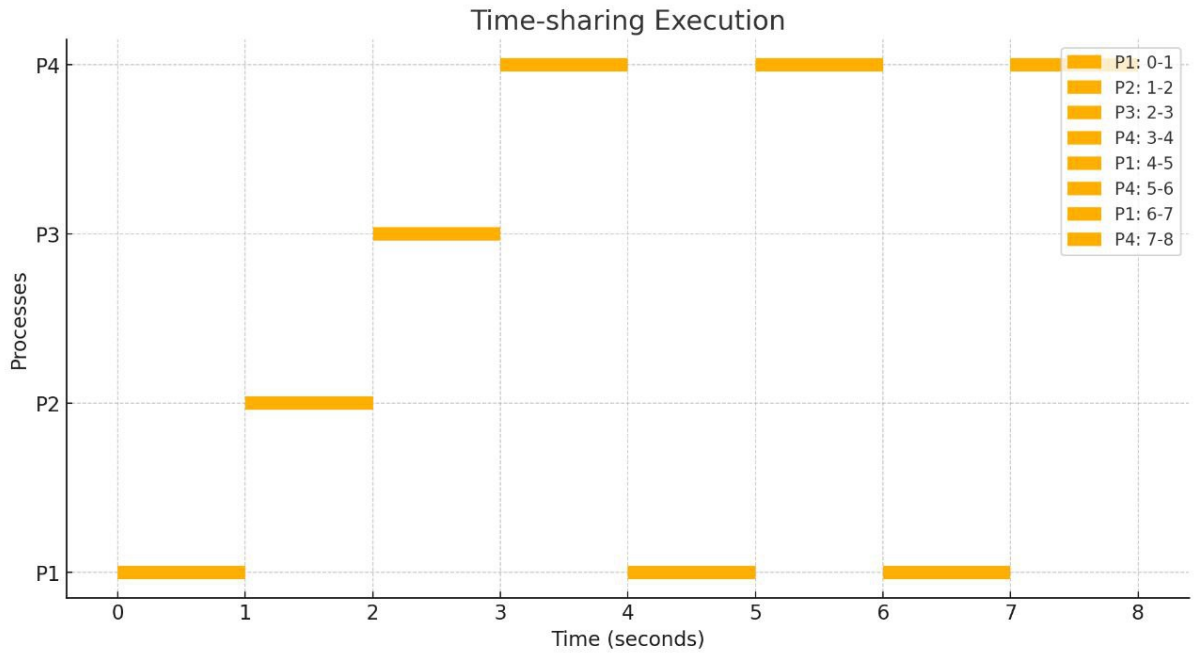
Tempo (s) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Processo | P1| P2| P3| P4| P1| P4| P1| P4| |

In questo diagramma, ogni processo esegue per 1 secondo in ordine ciclico fino al completamento. Questo metodo garantisce che nessun processo monopolizzi la CPU e che tutti i processi ricevano un'attenzione equa, migliorando l'interattività del sistema.

Rappresentiamo ora tutti questi casi come richiesto dalla consegna:





Multi tasking e time sharing impiegano lo stesso tempo ed eseguono i processi nello stesso ordine con un sampling di 1s

Il **"tempo di esecuzione dopo attesa"** si riferisce alla parte residua di lavoro che un processo deve svolgere dopo essere stato bloccato in attesa di un'operazione di input/output (I/O). Quando un processo effettua un'operazione I/O, di solito deve fermarsi, liberando la CPU. Una volta completata l'attesa (ad esempio, la lettura da

disco o una risposta da rete), il processo può tornare a occupare la CPU e terminare la sua esecuzione.

Questi tempi sono fondamentali per i **meccanismi di scheduling**, poiché il tempo di attesa I/O introduce **pause durante l'esecuzione**. Lo scheduler deve gestire l'uso della CPU ottimizzando queste pause, ad esempio:

- In **multi-tasking** o **time-sharing**, la CPU può essere assegnata ad altri processi durante l'attesa I/O.
- In **mono-tasking**, invece, il sistema attende il completamento dell'I/O, rendendo il tempo complessivo più lungo.

Come prevedibile, il multi tasking e il time sharing sono più efficienti ed efficaci