Scheduling

> ***DEF:*** *Lo scheduler dei processi è quel* ***componente del sistema operativo*** *che si occupa di* ***decidere quale processo va mandato in esecuzione****. Nell'ambito della multiprogrammazione è pensato per* ***mantenere la CPU occupata il più possibile****.*

> ***E.g:*** *Ad esempio avviando un processo mentre un altro è in attesa del completamento di un'operazione di I/O.*

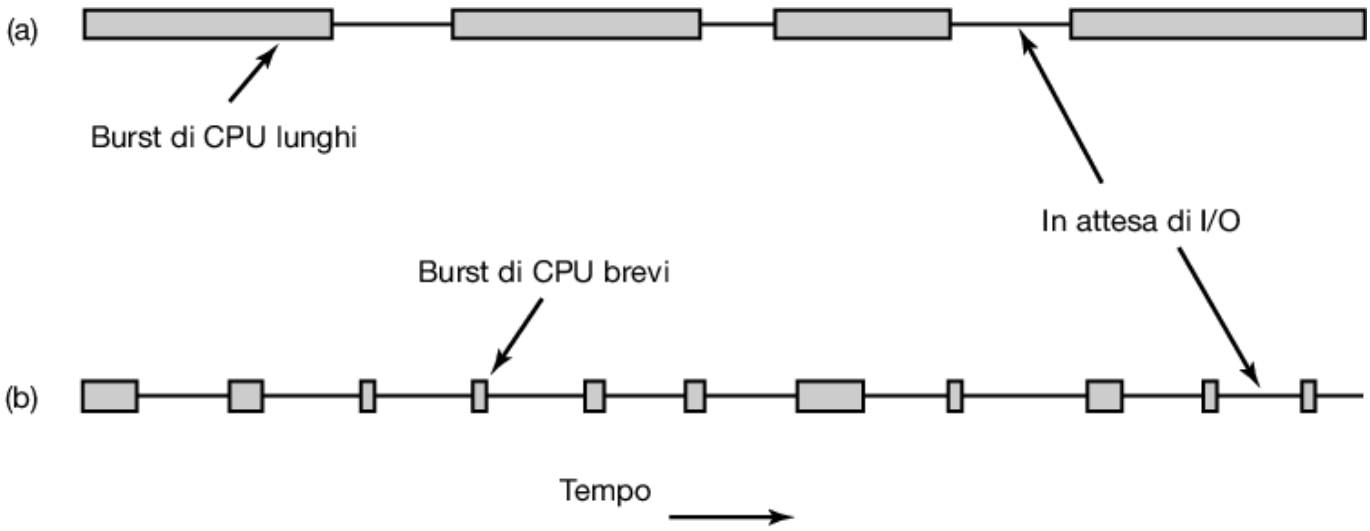
Comportamento tra processi

Durante la sua attività, un processo generalmente alterna le seguenti fasi:

> `CPU burst`: fase in cui viene **impiegata soltanto la CPU senza I/O**;

> `I/O burst`: fase in cui il processo **effettua input/output da/verso una risorsa (dispositivo) del sistema**.

> ***DEF:*** `burst` **(raffica)** *indica una sequenza di istruzioni.*



Alcuni processi, come quelli nella **Figura** `a` **spendono** la maggior parte del loro **tempo in elaborazione (uso CPU)**, mentre altri, come quelli nella **Figura** `b`, **spendono** la maggior parte del loro **tempo in attesa dell'I/O.**

La prima situazione è chiamata `compute-bound` **o** `CPU-bound`**, la seconda** `I/O-bound`**.**

> ***DEF:*** *I processi* `compute-bound` *hanno in genere* ***burst di CPU lunghi e attese di I/O poco frequenti***

> ***DEF:*** *I processi* `I/O-bound` ***hanno burst di CPU brevi e attese di I/O frequenti.***

> ***NOTA:*** *Quando un processo esegue* `I/O burst`, ***non utilizza la CPU****. In un sistema multiprogrammato,* ***lo scheduler assegna la CPU a un nuovo processo****.*

Categorie Algortimi di Scheduling

Gli algoritmi di scheduling si possono classificare in **due categorie**:

> **Con prelazione (`pre-emptive`):** ***al processo in esecuzione può essere revocata la CPU****. Il SO può, in base a determinati criteri,* ***sottrarre ad esso la CPU per assegnarla ad un nuovo processo.***

> **Senza prelazione (`non pre-emptive`):** ***la CPU rimane allocata al processo in esecuzione fino a quando esso si sospende volontariamente*** *(ad esempio, per I/O),* ***o termina****.*

Parametri dello Scheduling

Per analizzare e confrontare i diversi algoritmi di scheduling, si considerano i seguenti parametri, i primi tre relativi ai processi e gli altri relativi al sistema:

> **Tempo di attesa (`Wait time`):** *è la* ***quantità di tempo che un processo trascorre nella coda di pronto, in attesa della CPU.***

> **Tempo di completamento (`Turnaround time`)**: *è* ***l'intervallo di tempo che passa tra l'avvio del processo e il suo completamento.***

> **Tempo di risposta:** *è* ***l'intervallo di tempo tra avvio del processo e l'inizio della prima risposta.***

> **Utilizzo della CPU:** *esprime la* ***percentuale media di utilizzo della CPU nell'unità di tempo****.*

> **Produttività (`Throughput rate`):** *esprime il n****umero di processi completati nell'unità di tempo.***

Generalmente i parametri che devono essere massimizzati sono:

> **Utilizzo della CPU (al massimo: 100%)**

> **Produttività (`Throughput`)**

Invece, devono essere minimizzati:

> **Tempo medio di completamento (`Turnaround`) (soprattutto nei sistemi batch)**

> **Tempo medio di attesa**

> **Tempo medio di risposta (soprattutto nei sistemi interattivi)**

`[!] Non è possibile ottenere tutti gli obiettivi contemporaneamente.`

Categorie dei Sistemi dello Scheduler

Inoltre poiché differenti aree di applicazione (e differenti sistemi operativi) hanno obiettivi diversi. In altre parole, quello che lo scheduler dovrebbe ottimizzare non è la medesima cosa in tutti i sistemi.

Tre sistemi che vale la pena distinguere sono i seguenti: `batch`, `interattivo`, `real-time`.

```
Sistemi Batch
```

> Sistemi che fanno prevalentemente calcolo (**uso CPU massimo e massimo throughput**) e non fanno quasi **mai operazioni di I/O** e **minimizzano il tempo medio di completamento**.

`[i] DESCRIZIONE OPZIONALE.`

I sistemi batch sono ancora diffusamente usati nel mondo degli affari per l'elaborazione di paghe, inventari, esecuzione di accrediti e addebiti su conti corrente, calcolo degli interessi bancari, gestione dei reclami (nelle compagnie assicurative) e altre attività periodiche.

Nei sistemi batch non ci sono utenti impazienti in attesa al loro terminal di una risposta rapida a una piccola richiesta. Di conseguenza sono spesso accettabili algoritmi non preemptive o algoritmi preemptive con lunghi periodi per ciascun processo. Un simile approccio riduce gli scambi di processo e migliora così le prestazioni.

Gli algoritmi batch sono effettivamente abbastanza generali e spesso applicabili ugualmente ad altre situazioni, il che fa sì che valga la pena studiarli, anche da parte di persone non necessariamente coinvolte in elaborazioni di mainframe aziendali.

```
Sistemi Interattivi
```

> Qui si tiene a **minimizzare il tempo medio di risposta e attesa dei processi**.

`[i] DESCRIZIONE OPZIONALE.`

In un ambiente con utenti interattivi, l'uso della preemption è essenziale per evitare che un processo monopolizzi la CPU e neghi il servizio agli altri. Anche se intenzionalmente nessun processo viene eseguito

all'infinito, a causa di un errore di programma un processo potrebbe escludere tutti gli altri indefinitamente. La preemption è necessaria per prevenire questi comportamenti. Anche i server ricadono in questa categoria, dato che normalmente servono molteplici utenti (remoti), e tutti sempre di fretta.

```
Sistemi Real-Time
```

> Nei sistemi real-time, l'algoritmo con revoca non sempre è necessario dato che i **programmi real-time sono progettati per essere eseguiti per brevi periodi di tempo e poi si bloccano**. Inoltre si tende di più ad avere il **Rispetto delle scadenze e Prevedibilità**.

---

[i] DESCRIZIONE OPZIONALE.

In sistemi con vincoli real-time, la preemption è, stranamente, non sempre necessaria, poiché i processi sanno che non possono essere eseguiti per lunghi periodi di tempo e generalmente fanno il loro lavoro e si bloccano rapidamente. La differenza con i sistemi interattivi è che i sistemi real-time eseguono solo programmi destinati a promuovere l'applicazione a portata di mano. I sistemi interattivi sono generalistici e possono eseguire arbitrariamente programmi che non cooperano o che addirittura mirano a fare danni.

Riassunto

Tutti i sistemi

Equità – dare a ogni processo una equa condivisione della CPU

Applicazione della policy – assicurarsi che la policy dichiarata sia attuata

Bilanciamento – tenere impegnate tutte le parti del sistema

Sistemi batch

Throughput – massimizzare il numero di job per ora

Tempo di turnaround – ridurre al minimo il tempo da quando un lavoro è sottoposto alla sua fine

Utilizzo della CPU – mantenere la CPU sempre impegnata

Sistemi interattivi

Tempo di risposta – rispondere alle richieste rapidamente

Adeguatezza – far fronte alle aspettative dell'utente

Sistemi real-time

Rispetto delle scadenze – evitare la perdita di dati

Prevedibilità – evitare il degrado della qualità nei sistemi multimediali

Principali algoritmi di scheduling

Alcuni algoritmi sono usati sia nei sistemi batch che nei sistemi interattivi.

```
FCFS (First Came First Served)
```

Questo è l'algoritmo più semplice.

La CPU è assegnata ai processi seguendo l'ordine con cui essa è stata richiesta**, ovvero la cpu è assegnata al processo che è in attesa da più tempo nello stato di pronto.

Quando un processo acquisisce la CPU, resta in esecuzione fino a quando si blocca volontariamente o termina.

Quando un processo in esecuzione si blocca si seleziona il primo processo presente nella coda di pronto.

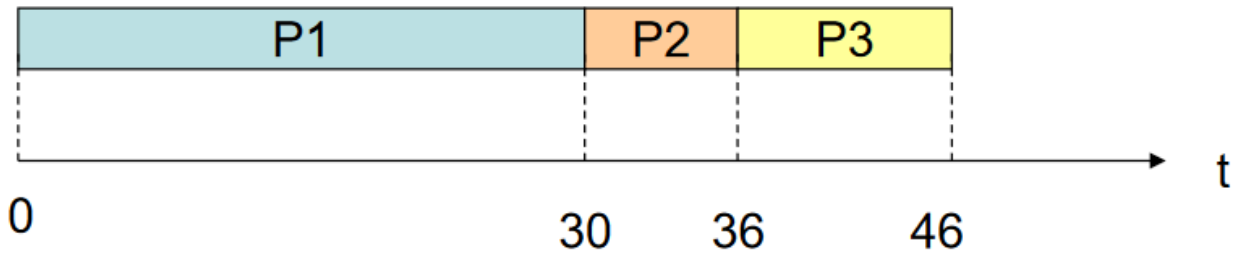
Quando un processo da bloccato ritorna pronto, esso è accodato nella coda di pronto.

E' inefficiente nel caso in cui ci sono molti processi che si sospendono frequentemente o nel caso di sistemi interattivi. Il tempo di attesa risulterebbe troppo lungo.

E' un algoritmo senza diritto di prelazione.

> **E.g:** Calcoliamo il tempo medio di attesa di tre processi `P1`, `P2` e `P3` avviati allo stesso istante e aventi rispettivamente i tempi (in millisecondi) di completamento pari a: `T1`**=30,** `T2`**=6 e** `T3`**=10**.

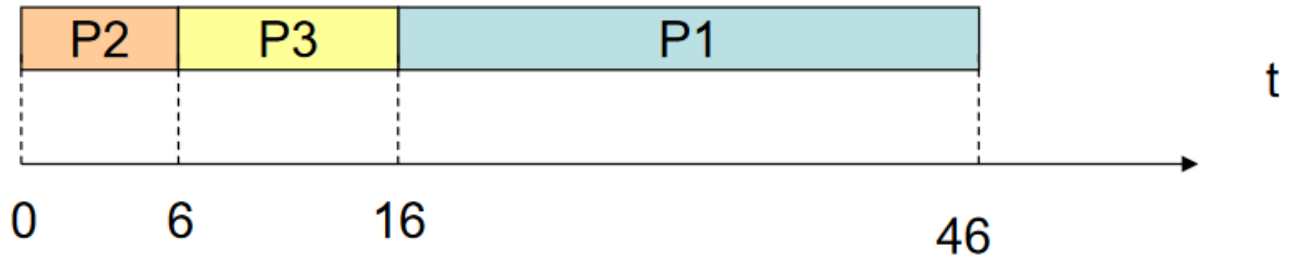
Esempio sui processi



Il tempo di attesa medio = (0+30+36)/3= 22.

Se cambiassimo l'ordine di scheduling nel seguente: {`P2,P3, P1`} il tempo medio di attesa passerebbe da **22 a 7,33**.

Esempio Cambio Ordine Processi

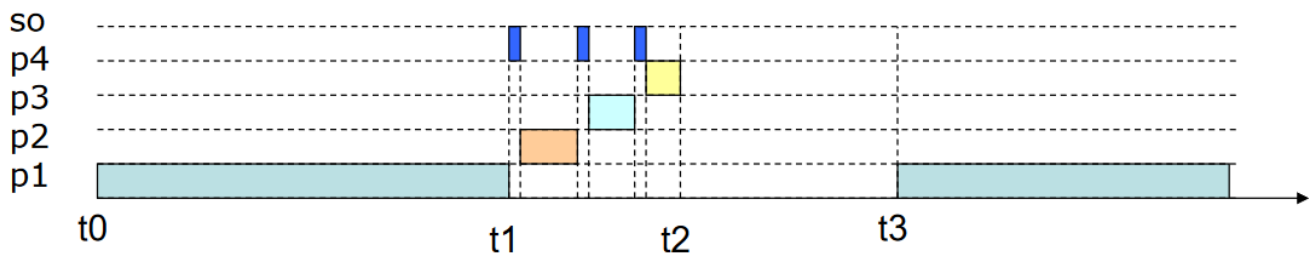


Ma il FCFS non consente di cambiare l'ordine dei processi.

Quindi nel caso in cui ci siano processi in attesa dietro a processi con lunghi CPU burst (processi CPU bound), il tempo di attesa sarà alto.

Inoltre, si ha la possibilità che si verifichi il cosiddetto ***effetto convoglio*** che si ha se molti processi I/O bound seguono un processo CPU bound. **Questa situazione porta ad basso livello di utilizzo della CPU.**

Esempio Effetto Convoglio



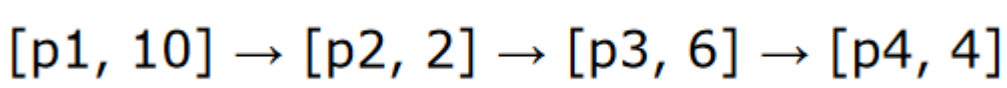
```
Shortest job first (SJF)
```

Gli algoritmi SJF, **possono essere sia non-preemptive (SNPF) sia preemptive (SRTF)**.

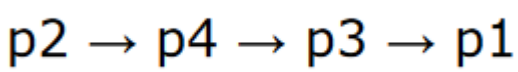
```
Shortest Next Process First (SNPF)
```

L'algoritmo SNPF prevede che sia **eseguito sempre il processo con il tempo di esecuzione più breve** tra quelli pronti.

> E.g: Supponiamo ad esempio che si trovino nello stato di pronto i seguenti processi, con la rispettiva durata di esecuzione in millisecondi:



Di conseguenza con SNPF, l'ordine dei processi diventa così.



P2 non attende nulla, perché va subito in esecuzione, P4 attende 2 millisecondi, perché va in esecuzione subito dopo P2, quindi P3 attende 6 millisecondi e P1 ne attende 12.

Il tempo di attesa medio è pari a:

Tempo attesa medio = **(0+2+6+12)/4 = 5 millisecondi**

```
Shortest Remaining Time First (SRTF)
```

L'algoritmo SRTF è la versione preemptive del precedente.

Con SRTF, se un **nuovo processo**, entrante nella coda di pronto, **ha una durata minore del tempo restante al processo in esecuzione** per portare a terminare il proprio CPU-burst, allora **lo scheduler provvede ad effettuare un cambio di contesto e assegna l'uso della CPU al nuovo processo.**

Ovvero, quando arriva un nuovo processo nella Ready List viene stimato il tempo di esecuzione che rimane al processo in possesso della CPU e lo si confronta con il tempo previsto del nuovo processo; **se quest'ultimo è minore si effettua il cambio di contesto** mandando in esecuzione il nuovo processo.

Esempio:

Esempio Lista Processi prima SRTF

Processi	Tempo di arrivo	Tempo esecuzione	Tempo di attesa	Istante iniziale	Istante finale	Turnaround time
P1	0	3		0	3	
P2	2	6		3	15	
P3	4	4		4	8	
P4	6	5		15	20	
P5	8	2		8	10	

All'istante 0 il processo P1 viene messo nello stato di READY e, **non essendoci altri processi ad occupare la CPU viene mandato in esecuzione.**

P2 viene inserito nella Ready List all'istante 2, con 6 istanti d'esecuzione.

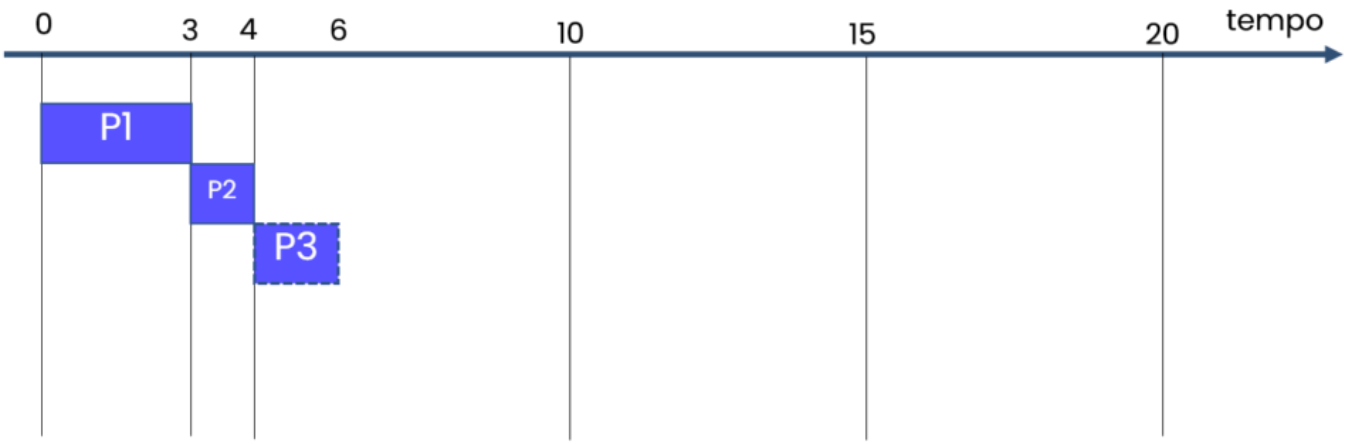
Questo dato viene confrontato con il tempo di esecuzione rimanente a P1, ovvero 1 istante (3 – 2) che essendo minore di 6 **non fa scaturire il cambio di contesto**.

P2 va in esecuzione all'istante 3, quando P1 termina (senza nessun confronto, visto che non ci sono altri processi nella Ready List).

All'istante 4 arriva in Ready List il processo P3, con tempo di esecuzione pari a 4, **un tempo minore rispetto a quello rimanente di P2** (ovvero 5, 6-1).

In questo caso **il sistema operativo procede con il cambio di contesto assegnando quindi la CPU a P3. P2 torna in Ready List con un tempo di esecuzione pari a 5.**

Lista Processi con scambio d'ordine

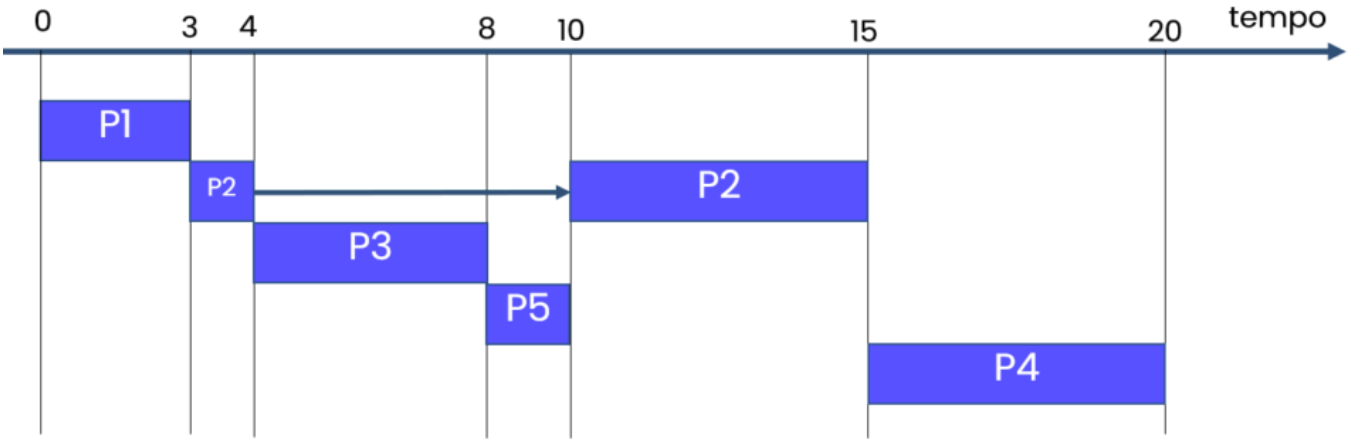


Ready List = {P2:5, P4:5}

Questo procedimento continua **finchè i processi precedenti non finisco e vengono ri-eseguiti quelli che erano stati rimessi nella Ready List** (P2 e P5).

Alla fine si avrà questo risultato.

`Lista Processi dopo SRTF`




```
Round Robin
```

E' stato realizzato **per i sistemi time-sharing**.

Consente il prerilascio e utilizza la modalità FIFO. (ovvero una lista in cui i processi sono ordinati **secondo l'ordine di arrivo**)

Al **primo processo in coda viene assegnata la CPU per un periodo di tempo prefissato** (`time slice`).

Se il processo termina prima della fine del quanto di tempo **lascia volontariamente la CPU** (che verrà assegnata al nuovo primo processo in coda), **se non è così viene rimosso** dal sistema operativo **e posto in fondo alla coda** (anche in questo caso l'esecuzione passerà al primo processo nella coda).

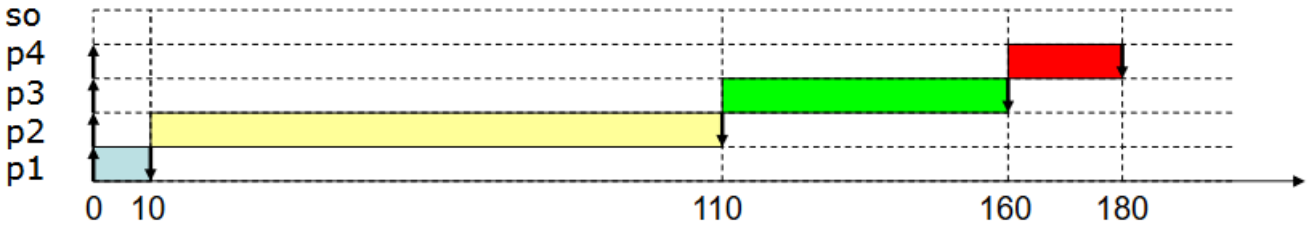
Esempio:

Supponiamo che all'istante T0 siano presenti i seguenti quattro processi nella coda di pronto. Calcoliamo il tempo medio di attesa e di risposta per quattro processi P1, P2, P3 e P4 aventi rispettivamente i tempi di arrivo e durata di CPU-burst (in millisecondi) pari a:

```
P1 (10ms), P2 (100ms), P3 (50ms), P4 (20ms)
```

```
    Usando FCFS
```

Intervallo di tempo usando FCFS



Usando FCFS si hanno questi valori:

Processo	Tempo di Attesa	Tempo di Risposta
P1	0	10
P2	10	110
P3	110	160
P4	160	180

Quindi:

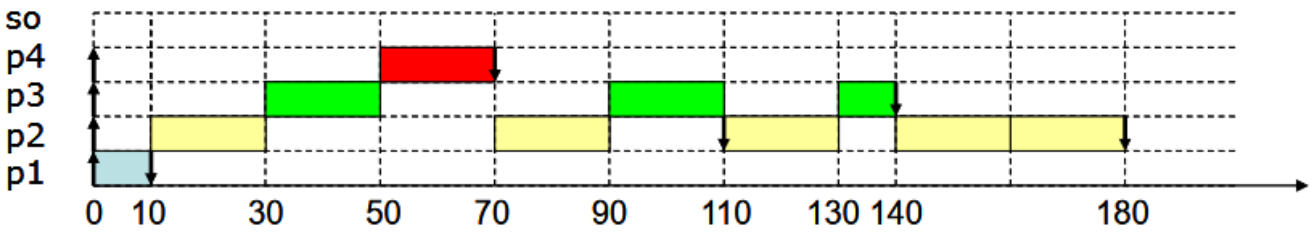
> **Attesa Medio** = (P1 + P2 + P3 + P4)/nP = **70**

> **Risposta Medio** = (P1 + P2 + P3 + P4)/nP = **115**

```
Usando RR
```

Viene impostato un quanto di tempo (Δ) = `20 ms` si ha questo instante:

Intervallo di tempo usando RR



Usando RR si hanno questi valori:

Processo	Tempo di Attesa	Tempo di Risposta

Processo	Tempo di Attesa	Tempo di Risposta
P1	0	10
P2	(10)+(40)+(20)+(10)	180
P3	(30)+(40)+(20)	140
P4	50	70

Quindi

> **Attesa Medio** = (P1 + P2 + P3 + P4)/nP = **55**

> **Risposta Medio** = (P1 + P2 + P3 + P4)/nP = **100**