Sistema Operativo
Gestione dei processi
Algoritmi di scheduling

Unità 4 – Lezione 3 – Gestione del processore

Processi CPU bound e I/O bound

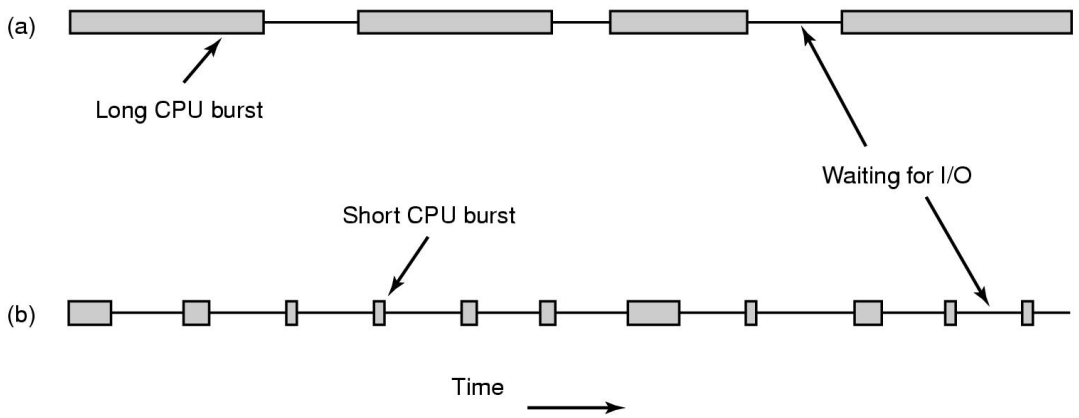
- L'esecuzione di un processo consiste in fasi di elaborazione CPU e fasi di attesa I/O
- Il processi possono essere caratterizzati come
  - CPU bound: maggior parte del tempo speso usando la CPU
  - I/O bound: maggior parte del tempo speso ad attendere operazioni di I/O

Obiettivo

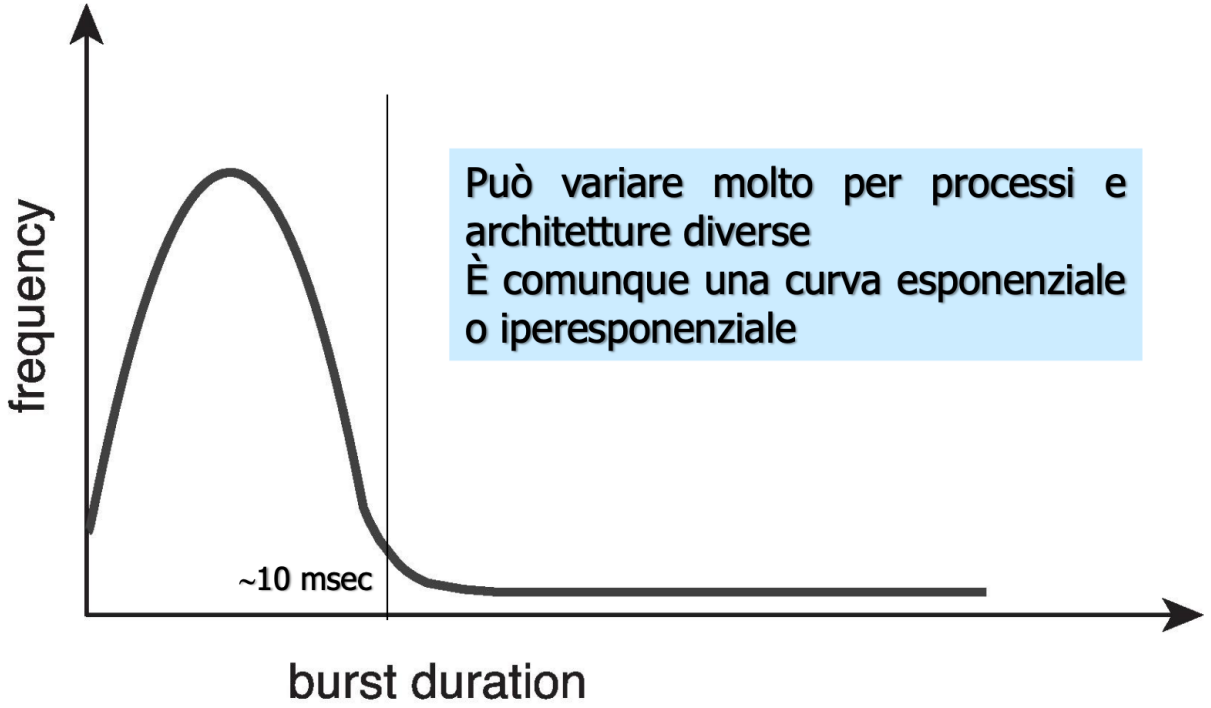
avere sempre processi

presenti nelle code dei processi

pronti e dei dispositivi

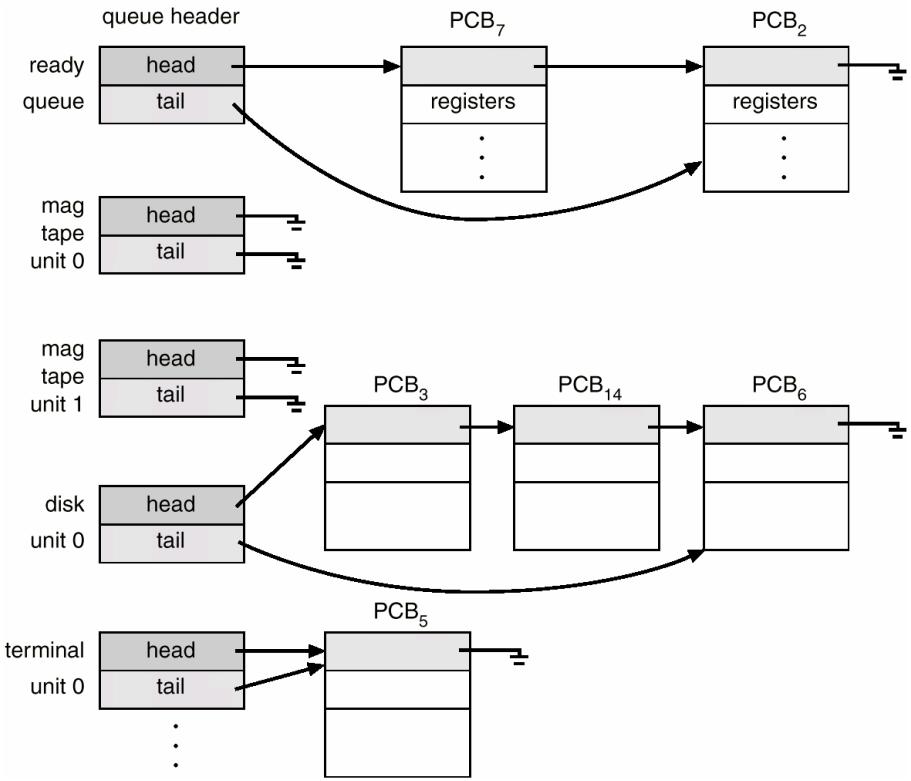


Durata operazioni CPU e frequenza



Può variare molto per processi e architetture diverse
È comunque una curva esponenziale o iperesponenziale

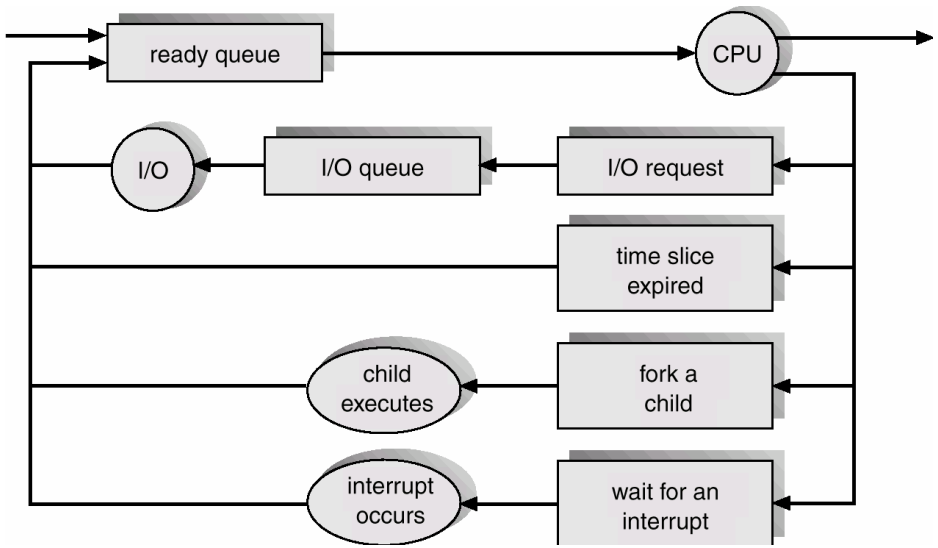
PCB e code



- Multiprogrammazione: impiegare al massimo la CPU tenendo in memoria numerosi processi in attesa di essere eseguiti
- Abbiamo detto che SO mantiene diverse code, liste di PCB per i processi attivi
  - Le PCB dei processi si spostano tra le varie code
- I processi che non sono in CPU sono in una coda
- Context switch, overhead di sistema
- Dispatcher:
  - Cambio di contesto
  - Passaggio user mode
  - Salto posizione programma utente per riavviare esecuzione

Scheduler

- Lo scheduler è la parte del SO che decide quale tra i processi in **ready queue** deve utilizzare la CPU (scheduler a breve termine)
  - Deve essere chiamato molto spesso (circa 100msec)
  - Deve essere veloce (circa 1 msec)
- Ci sono altri tipi di scheduler
  - A lungo termine: seleziona quali processi vengono caricati dalla memoria secondaria
  - A medio termine: rimuove processi da memoria e da contesa CPU facilitando compito scheduler breve termine



Pre-emptive e non

1. Da stato running a wait (e.g., richiesta I/O o richiesta attesa terminazione figlio)
2. Da stato running a ready (e.g., segnale interrupt)
3. Da wait a ready (e.g., fine operazione I/O)
4. Processo termina

Casi 1 e 4 non comportano scelta di scheduling e sono senza prelazione

- Uno scheduler può operare in maniera
  - Pre-emptive (con prelazione)
    - Processi rimossi da CPU
    - Essenziale per SO interattivi e time-sharing general purpose
    - Costosta per I cambi di contesto
    - Anche più complicato da gestire (e.g., condivisione dati tra processi)
    - Si ripercuote anche su progettazione nucleo
  - Non preemptive (senza prelazione)
    - I processi mantengono uso CPU fino al loro completamento
    - Processi poco importanti possono far ritardare quelli importanti
    - Sistemi batch e spesso anche real-time

Obiettivi di scheduling

- **Max. utilizzo CPU (% CPU, Burst CPU)**
- **Max. Throughput** produttività del sistema, quanti processi sono completati nell'unità di tempo
- **Min. Turnaround Time** tempo di completamento, quanto passa da quando processo ha iniziato esecuzione e da quando la completa (somma dei tempi passati in attesa nelle code, in esecuzione e in operazioni I/O)
- **Min. Tempo attesa** tempo speso in attesa nella ready queue. Si ottiene dalla somma degli intervalli di attesa
- **Min. Tempo risposta** tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta

***Nota:** si ottimizzano in genere i valori medi, in alcuni casi i valori minimi o massimi

Obiettivi di scheduling

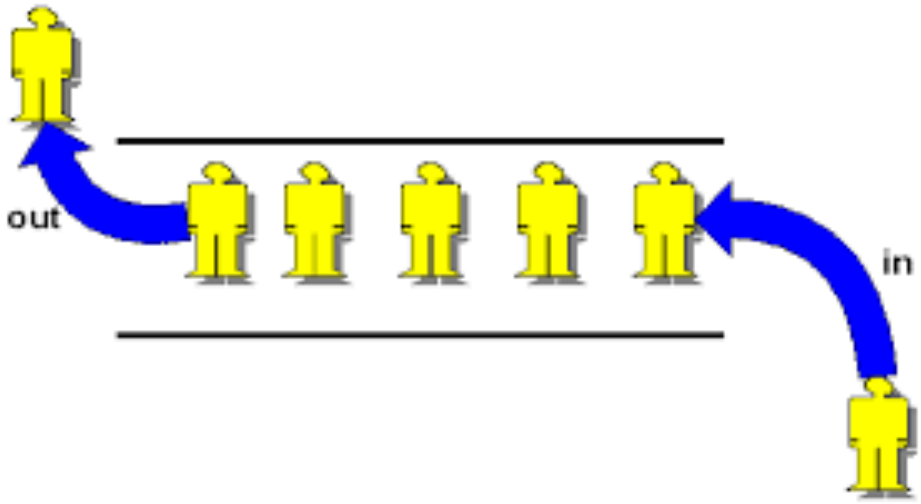
- Concetto di fairness: equità (dare ad ogni processo una prozione equa della CPU)
- Bilanciamento (tenere occupate tutte le parti del sistema)
- Uso della CPU (tenere sempre occupata la CPU)
- Politiche di controllo, verificare che le politiche siano messe in atto
- **Nota:** sistemi differenti possono avere obiettivi differenti
  - Sistemi batch
    - Throughput (massimizzare job per unità di tempo)
    - Turnaround time (minimizzare il tempo di esecuzione)
  - Sistemi interattivi
    - Response time (minimizzare I tempi di risposta)
  - Sistemi Real-time
    - Rispettare le scadenze

Algoritmi di Scheduling

- Iniziamo a vedere gli algoritmi di scheduling
- Visione semplicata
  - Negli esempi ed esercizi si considera una sola sequenza di operazioni della CPU (durata epressa in millisecondi)
  - Le misure di confronto sono tempo di attesa medio e tempo di completamento medio ma ci sono misure più complicate

FCFS – First Come First Served

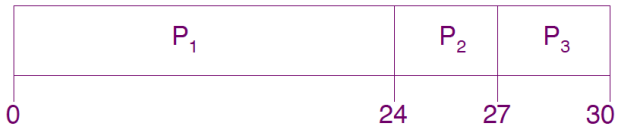
- Il più semplice, algoritmo di scheduling in ordine di arrivo
- Non preemptive (senza prelazione)
  - Un processo lascia la CPU se termina o se chiede I/O
- Coda FIFO (First In First Out)
  - Quando un processo entra nella code dei procesi pronti si collega il suo PCB all'ultimo elemento della code
  - La CPU viene assegnata al processo il cui PCB si trova in testa alla coda



FCFS esempio

Processo	Tempo di burst
P_1	24
P_2	3
P_3	3

- I processi arrivano al sistema nell'ordine: P_1 , P_2 , P_3.
  Il diagramma di Gantt per lo scheduling **FCFS** è:



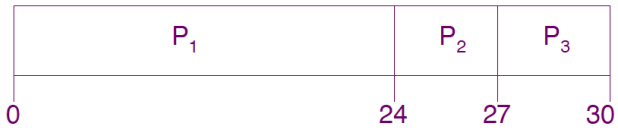
- Tempi di attesa: P_1 = 0; P_2 = 24; P_3 = 27.
- Tempi di completamento: P_1 = 24; P_2 = 27; P_3 = 30.
- Tempo medio di attesa = (0 + 24 + 27)/3 = 17.
- Tempo medio di competamento = (24 + 27 + 30)/3 = 27.

Può un tempo di attesa medio molto lungo che dipende dall'ordine di arrivo dei processi

FCFS esempio

Processo	Tempo di burst
P_1	24
P_2	3
P_3	3

- I processi arrivano al sistema nell'ordine: P_1 , P_2 , P_3.
  Il diagramma di Gantt per lo scheduling **FCFS** è:



- Tempi di attesa: P_1 = 0; P_2 = 24; P_3 = 27.
- Tempi di completamento: P_1 = 24; P_2 = 27; P_3 = 30.
- Tempo medio di attesa = (0 + 24 + 27)/3 = 17.
- Tempo medio di competamento = (24 + 27 + 30)/3 = 27.

Può un tempo di attesa medio molto lungo che dipende dall'ordine di arrivo dei processi

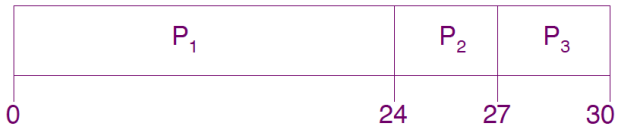
Se ordine di arrivo è P_2, P_3 e P_1

1. Create diagramma di Gantt
2. Calcolate i tempi medi di attesa e completamento

FCFS esempio

Processo	Tempo di burst
P_1	24
P_2	3
P_3	3

- I processi arrivano al sistema nell'ordine: P_1 , P_2 , P_3.
  Il diagramma di Gantt per lo scheduling **FCFS** è:



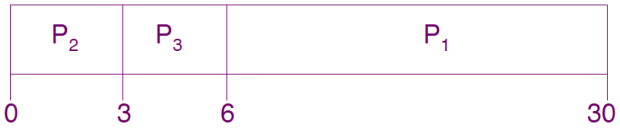
- Tempi di attesa: P_1 = 0; P_2 = 24; P_3 = 27.
- Tempi di completamento: P_1 = 24; P_2 = 27; P_3 = 30.
- Tempo medio di attesa = (0 + 24 + 27)/3 = 17.
- Tempo medio di competamento = (24 + 27 + 30)/3 = 27.

Può un tempo di attesa medio molto lungo che dipende dall'ordine di arrivo dei processi

Se l'ordine di arrivo è

P_2 , P_3 , P_1,

il diagramma di Gantt risulta…



- Tempi di attesa: P_1 = 6; P_2 = 0; P_3 = 3.
- Tempi di completamento: P_1 = 30; P_2 = 3; P_3 = 6.
- Tempo medio di attesa = (6 + 0 + 3)/3 = 3.
- Tempo medio di completamento = (30 + 3 + 6)/3 = 13.

- Non si verifica l'effetto, per cui processi di breve durata devono attendere che un processo molto lungo liberi la CPU.

FCFS considerazione (1)

Caratteristiche

- il tempo di attesa è basato sui job CPU bound
- l'utilizzo della CPU può essere bassa (se non sono tutti CPU bound)
- produce una bassa utilizzazione dei dispositivi di I/O
- Non adatto per i sistemi time-sharing perché non garantisce affatto interattività
- Va ancora peggio per i sistemi real-time, perché non è pre-emptive
- Potrebbe andare bene per i sistemi a lotti batch

FCFS considerazioni (2)

- Un processo CPU bound e molti I/O bound
  - Mentre il processo CPU bound gli altri completano le operazioni di I/O e si spostano nella ready queue. Tutti I processi sono in ready queue e i dispositivi di I/O sono inattivi
  - Il processo CPU bound si sposta a fare I/O. I process I/O bound usano CPU e sono in wait per I/O. La CPU rimane inattiva
  - Il processo CPU torna ad usare la CPU e di nuovo I/O dispositive inattivi

- Effetto convoglio:
  - Tutti I processi attendono che un lungo processo liberi la CPU causando una riduzione dell'uso di CPU e delle risorse (non buon bilanciamento)
  - Sarebbe meglio eseguire prima I processi più brevi

- Non ottimale per I sistemi time-sharing

Shortest-Job-First (SJF)

- Scheduling per brevità: si associa a ciascun processo la lunghezza del suo burst di CPU successivo. Si opera lo scheduling in base al **tempo rimanente dei processi** (i.e., mando in esecuzione chi dura meno)
- Due opzioni
  - Non pre-emptive
  - Pre-emptive se arriva un nuovo processo con burst di CPU minore del tempo rimasto per il proccesso corrente, il nuovo processo ha prelazione su CPU (detto **Shortest Remaining Time First – SRTF**)
- Algoritmo ottimale, rende minimo il tempo medio d'attesa

SJF esempio – non pre-emptive

NON PREEMPTIVE

Processo	Tempo di arrivo	Tempo di burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive):

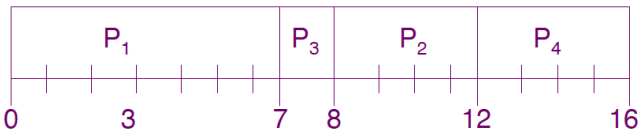
Nota: a parità di burst FCFS, P2 prima di P4

SJF esempio – non pre-emptive

NON PREEMPTIVE

Processo	Tempo di arrivo	Tempo di burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive):



- Tempo medio di attesa = (0 + 6 + 3 + 7)/4 = 4.
- Tempo medio di completamento = (7 + 10 + 4 + 11)/4 = 8.

Nota: a parità di burst FCFS, P2 prima di P4

SJF esempio preemptive

PREEMPTIVE

(shortest remaining time first)

Processo	Tempo di arrivo	Tempo di burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

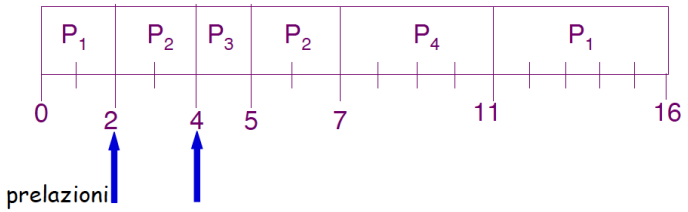
SJF esempio preemptive

PREEMPTIVE

(shortest remaining time first)

Processo	Tempo di arrivo	Tempo di burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive):



- Tempo medio di attesa = (9 + 1 + 0 +2)/4 = 3.
- Tempo medio di completamento = (16 + 5 + 1 +6)/4 = 7.

SJF esempio preemptive calcoli

P1 attende

tempo arrivo t0

eseguito per 2 millisecondi t0-t2: attesa 0

riprende a t11: attesa 9 (t11 – t2)

P2 attende

tempo arrivo t2

eseguito per 2 millisecondi t2 - t4: attesa 0

riprende a t5: attesa 1 (t5- t4)

P3 attende 0

P4 attende

tempo arrivo t5

eseguito per 4 millisecondi t7-t11: attesa 2 (t7-t5)

Completamento: tempo di fine – tempo di arrivo

P1 t16 – t0 = 16

P2 t7 – t2 = 5

P3 = t5-t4 = 1

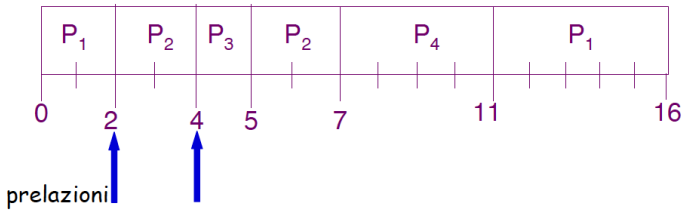
P4 = t11 – t5 = 6

PREEMPTIVE

(shortest remaining time first)

Processo	Tempo di arrivo	Tempo di burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive):



- Tempo medio di attesa = (9 + 1 + 0 +2)/4 = 3.
- Tempo medio di completamento = (16 + 5 + 1 +6)/4 = 7.

Esercizio in classe

Processo	Istante arrivo	Durata sequenza
P1	0	8
P2	1	4
P3	2	9
P4	3	5

- 4 processi
- Calcolare il tempo medio d’attesa e il tempo medio di completamento con FCFS SJF e SRTF

SJF e SRFT considerazioni

Problema:

- La durara della CPU non la posso calcolare, posso solo stimarla (media esponenziale usando i burst vecchi e recenti, pesando meno i vecchi e di più i recenti)

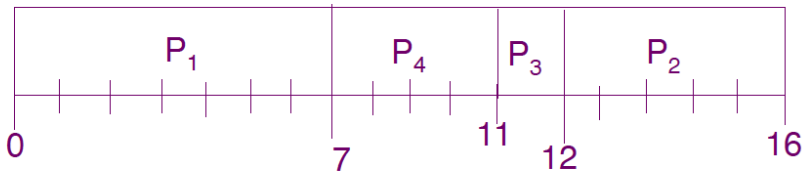
Scheduling con priorità

- L'algoritmo SJF è un caso particolare dell'algoritmo di scheduling con prioritá: si associa una priorità ad ogni processo
  - Se priorità uguale, FCFS
  - SJF: la priorità è rapprsentata dal successivo tempo di burst
  - Convenzione: numeri bassi prioritá bassa (Windows Linux)
- Lo scheduling con prioritá può essere con o senza prelazione
- La prioritá può derivare da:
  - Importanza processo, tipo limiti di tempo, requisiti memoria, numero file aperti, trapporto tra lunghezza media seuqenze operazioni I/O e operazioni CPU
  - Possono anche essere esterne: processi di un dipartimento eseguiti prima di altri etc.

Esempio priorità senza prelazione

p=0 massima priorita'

Processo	priorita'	Tempo di burst
P_1	1	7
P_2	3	4
P_3	2	1
P_4	1	4



- Tempo medio di attesa = (0 + 12 + 11 +7)/4 = 7.5.
- Tempo medio di completamento = (7 + 16 + 12 +11)/4 = 11.5

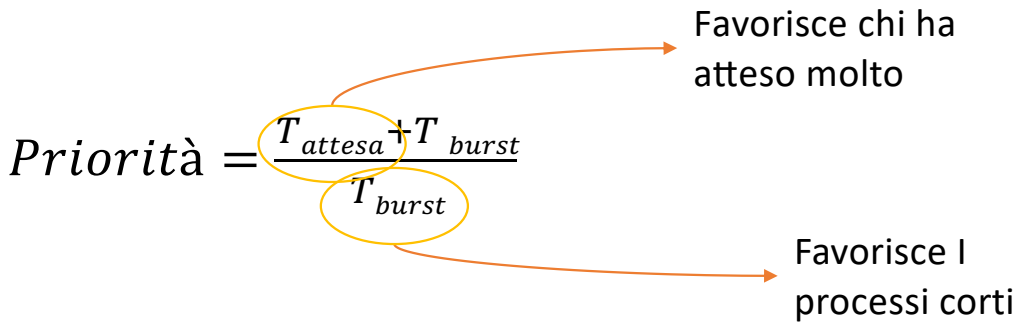
Starvation (attesa indefinita)

- Letteralmente… morte per inedia!
- Un algoritmo di scheduling con priorità darà sempre precedenza ai processi con priorità più alta se questi continuano ad arrivare. I processi con priorità bassa… muoiono di fame, non accedono mai alla CPU
- SJF: Processi molto lunghi tendono a non essere eseguii se continuano ad arrivare processi brevi… che succede?
  - Corre voce che, quando fu fermato l'IBM 7094 al MIT, nel 1973, si scoprì che un processo con bassa priorità sottoposto nel 1967, non era ancora stato eseguito

CHE SI FA?

Aging (invecchiamento)

- Aumento graduale della priorità dei processi che si trovano in attesa nel sistema da lungo tempo
- Priorità variabile e non fissa: la priorità si aggiorna dinamicamente in base a quanto un processo sta aspettando
- Per esempio, HRRNS (High Response Ratio Next Scheduling):



Round Robin – scheduling circolare

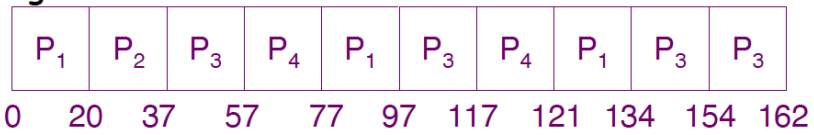
- Progettato appositamente per sistemi time-sharing
- A ciascun processo viene allocata una piccola unità di tempo di CPU (quanto di tempo o timeslice), 10-100 millisecondi.
- Trascorso il tempo, il processo è forzato a rilasciare la CPU e riaccodato nella ready queue (preemptive per definizione)
  - Se il processo finisce prima del quanto di tempo, rilascia volontariamente la CPU
- La ready queue è una coda FIFO
- Richiede un temporizzatore, un clock, che invia un segnale quando scade il tempo
- Con questo algoritmo tutti i processi sono trattati allo stesso modo, in una sorta di "correttezza" (fairness), e possiamo essere certi che non ci sono possibilità di starvation perché tutti a turno hanno diritto a utilizzare la CPU.

Round Robin esempio

Time slice 20 ms

Processo	Tempo di burst
P_1	53
P_2	17
P_3	68
P_4	24

- Il diagramma di Gantt è:



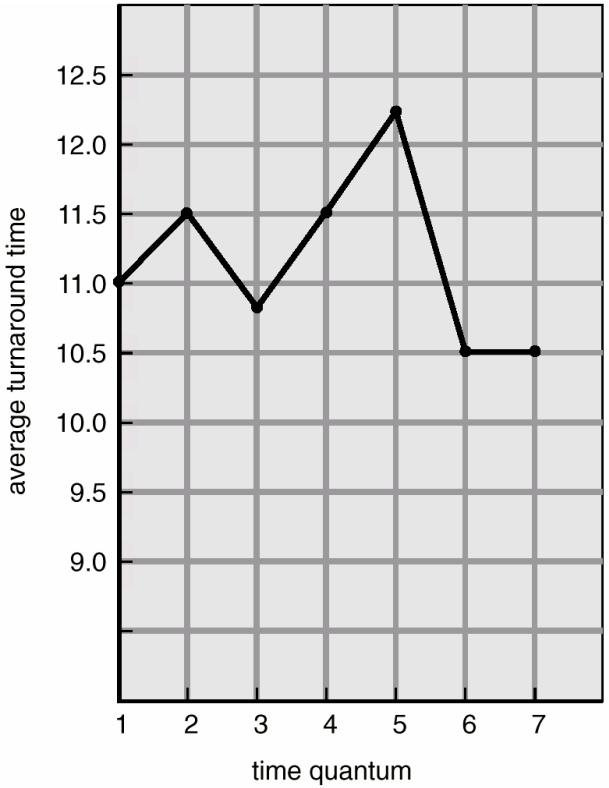
- In genere si ha un tempo medio di attesa maggiore rispetto a **SJF**, tuttavia si ha un miglior tempo di completamento per i processi lunghi.

Round Robin considerazioni

- Il dispositivo che rende possibile la sospensione di un processo ancora in esecuzione allo scadere del time slice, è il Real-time clock (RTC).
  - Esso non è altro che un chip impiantato sulla scheda madre contenente un cristallo di quarzo che viene fatto oscillare in modo estremamente stabile con segnali elettrici
  - tali oscillazioni scandiscono il tempo generando periodicamente delle interruzioni da inviare al sistema operativo.
- Importante dimensionare il time slice:
  - quando è piccolo abbiamo tempi di risposta ridotti ma è necessario effettuare frequentemente il cambio di contesto tra i processi, con notevole spreco di tempo e quindi di risorse (overhead);
  - quando è grande i tempi di risposta possono essere elevati e l'algoritmo degenera in quello di FCFS
- RR rischia di dare meno importanza a processi importanti

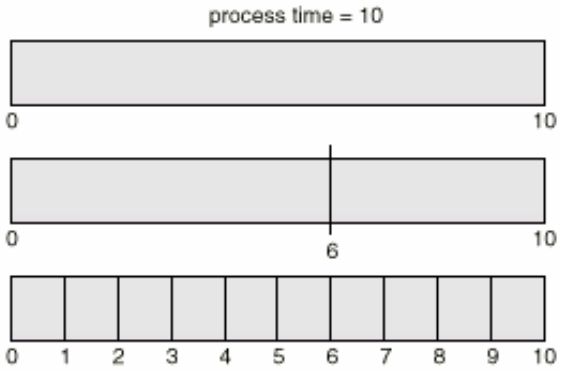
Round Robin – considerazioni time slice

Regola empirica: 80% dei CPU burst dovrebbe essere minore del quanto



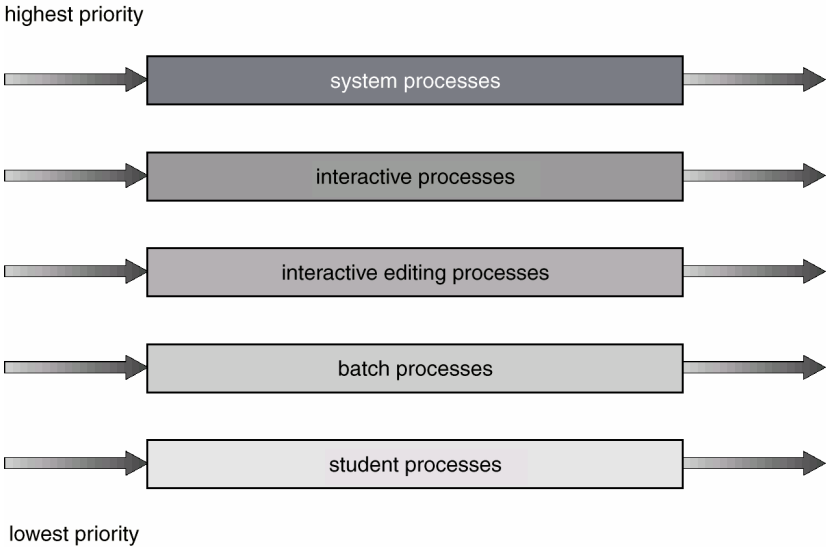
process	time
P_1	6
P_2	3
P_3	1
P_4	7

Quanto troppo piccolo, il Sistema impiega più tempo nei context switch



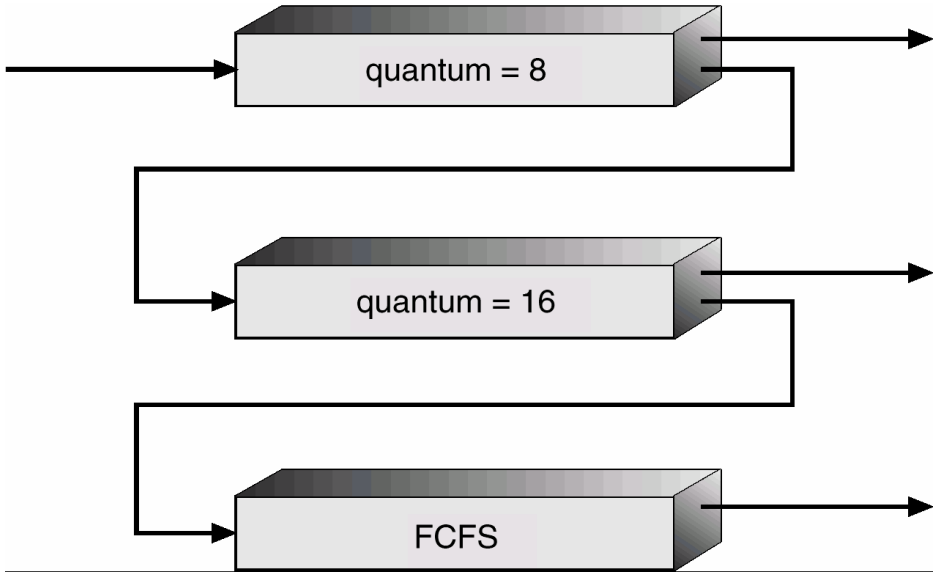
MLQS – Multiple Level Queue Scheduling con code multiple

- La ready queue è divisa in code separate:
  - Processi foreground (interattivi)
  - Processi background (batch)
- Ogni coda ha il suo algoritmo di scheduling
  - Foreground RR
  - Background FCFS
- + algoritmo di scheduling tra le code
  - Preemptive e priorità fissa
  - Ogni coda ha un burst CPU



MLFQS con retroazione

- Un processo può spostarsi tra le varie code
  - Numero di code
  - Algoritmi di scheduling per ciascuna coda
  - Metodo per spostare processo a coda con priorità maggiore
  - Metodo per spostare processo a coda con priorità minore
  - Metodo per determinare la coda iniziale di un processo



MLFQ con retroazione

- La definizione di uno scheduler a code multiple con retroazione (o feedback) costituisce il più generale criterio di scheduling della CPU che nella fase di progettazione si può adeguare a un sistema specifico
- E’ il più generale ma è anche il più complesso

Esempio priorità Windows

- Una ready-queue per ogni classe di processi
- Importanza differente
  - Alta (sistema o finestre foreground)
  - Bassa (finestre ridotto o I background)
  - Priorità variabile
    - Aumenta se in attesa di eventi
    - Diminuisce se finisce timeslice

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Esempio priorità Mac

- **Preemptive Multitasking & Time-Sharing**
  - Interrompe i processi per gestire quelli con priorità più alta.
  - Suddivide il tempo CPU in “quantum” e assegna turni ai processi.
  - Favorisce i processi interattivi rispetto a quelli in background.
- **Scheduler basato su priorità**
  - Classifica i processi in base a priorità dinamiche.
  - **Processi real-time**: Priorità più alta per compiti critici.
  - **Processi utente**: Variano in priorità, con maggiore attenzione a quelli interattivi.
- **Real-Time Scheduling**
  - Garantisce che processi cruciali ricevano risorse CPU in tempo.
- **Uso di nice**
  - Modifica la priorità dei processi (da -20 a +20).
  - **renice** per cambiare la priorità di processi già in esecuzione.

Sudo htop e nice

- nice è un comando utilizzato per modificare la **priorità di scheduling** di un processo. In macOS (così come in altri sistemi Unix-like), il valore **nice** rappresenta un suggerimento al sistema operativo su quanto "gentile" (ovvero quanto meno prioritario) dovrebbe essere il processo rispetto agli altri.
- Il valore di **nice** può andare da **-20** a **+20**:
- Un valore di **-20** rappresenta la **massima priorità** per il processo.
- Un valore di **+20** indica la **priorità minima** (il processo è meno competitivo per ottenere tempo CPU).

Approfondimento Windows e Linux

Conclusioni

- Molti altri algoritmi di scheduling
- Per esempio, sistemi real-time hanno algoritmi di scheduling differenti dai sistemi interrativi / time-sharing
- Ogni sistema operativo implementa una sua versione particolare degli algoritmi di scheduling
- Scheduling multiprocessore
  - Ancora più complesso, no soluzione ottima

Sincronizzazione tra processi

- I processi possono essere indipendenti o dover co-operare/competere/comunicare tra di loro
- Vedremo quali sono I metodi per
  - Comunicare tra processi
    - Variabili comuni / memoria condivisa
    - Scambio di messaggi
    - segnali
  - Sincronizzare I processi per co-operazione (uno produce, l'altro consuma)
  - Sincronizzare I processi perché competono per una risorsa