Processi e Thread

Scheduling

(Schedulazione)

Scheduling
Introduzione al problema dello Scheduling (1)

- Lo scheduler si occupa di decidere quale fra i processi *pronti* può essere mandato *in esecuzione*
- L'algoritmo di scheduling ha impatto su:
  - prestazioni percepite dagli utenti
  - efficienza nell'utilizzo delle risorse della macchina
- Lo scheduling ha obiettivi diversi in diversi sistemi (batch, interattivi…)

Introduzione al problema dello Scheduling (2)

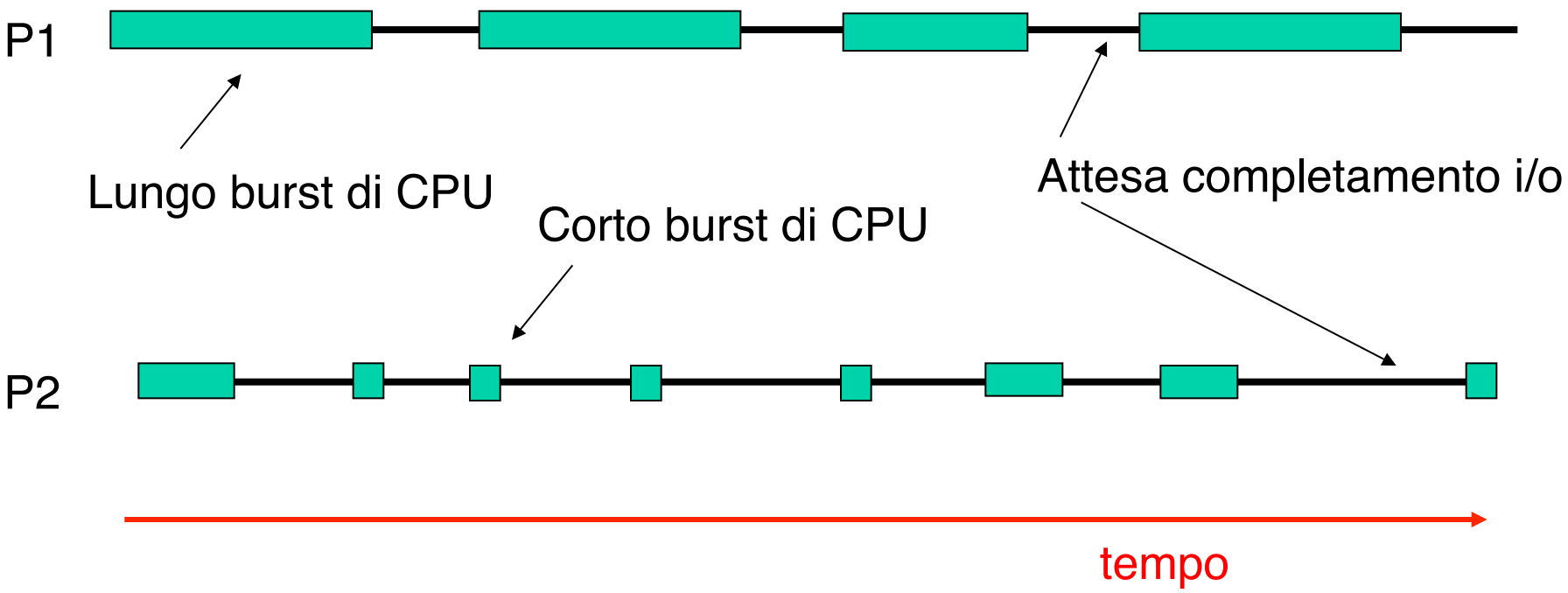
Obiettivi principali degli Algoritmi di Scheduling:

- *Fairness* (Equità) - processi della stesso tipo devono avere trattamenti simili
- *Balance* (Bilanciamento) - tutte le parti del sistema devono essere sfruttate (CPU, dispositivi …)
- Sistemi batch
  - *Throughput* - massimizzare il numero di job completati in un intervallo di tempo
  - *Tempo di Turnaround* - minimizzare il tempo di permanenza di un job nel sistema
- Sistemi interattivi
  - *Tempo di risposta* - minimizzare il tempo di riposta agli eventi
  - *Proporzionalità* - assicurare che il tempo di risposta sia proporzionale alla complessità dell'azione richiesta

Introduzione al problema dello Scheduling (3)

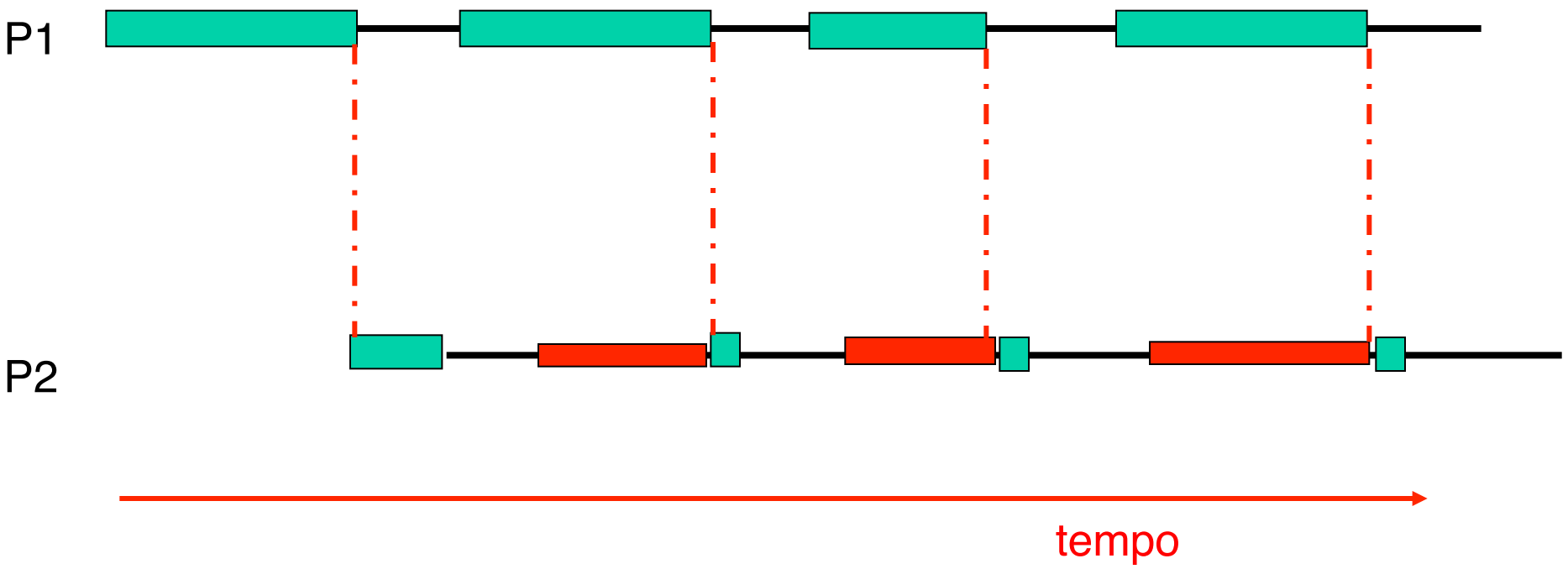
- Due tipologie di processi :
  - processi *CPU-bound* -- lunghi periodi di eleborazione fra due richieste successive di I/O
  - processi *I/O-bound* -- brevi periodi di elaborazione fra due richieste successive di I/O
- Conviene dare priorità ai processi *I/O-bound*

Introduzione al problema dello Scheduling (4)



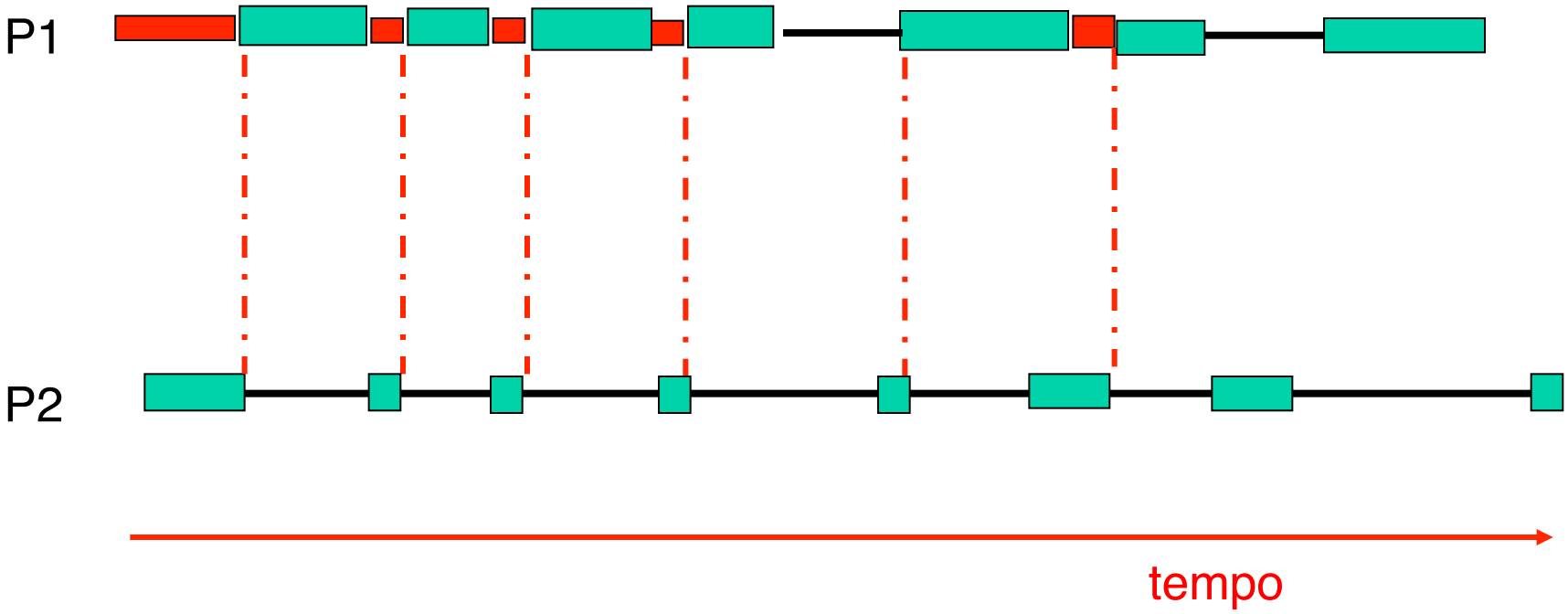
- Processi *compute bound* (P1) and *I/O bound* (P2)

Introduzione al problema dello Scheduling (5)



- Priorità ai *compute bound*

Introduzione al problema dello Scheduling (6)



- Priorità agli I/O *bound*
  - il funzionamento del sistema è più bilanciato

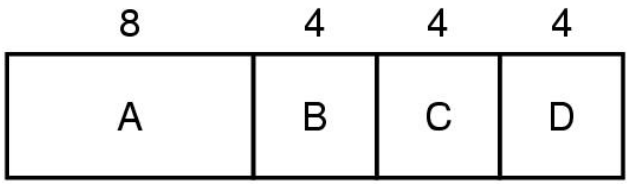
Introduzione al problema dello Scheduling (7)

- Scheduling *senza prerilascio*
  - lo scheduler interviene solo quando un processo viene creato, termina o si blocca su una SC
- Scheduling *con prerilascio*
  - lo scheduler può intervenire ogni volta che è necessario per ottenere gli obiettivi perseguiti
    - quando diventa pronto un processo a più alta priorità rispetto a quello in esecuzione
    - quando il processo in esecuzione ha sfruttato la CPU per un tempo abbastanza lungo

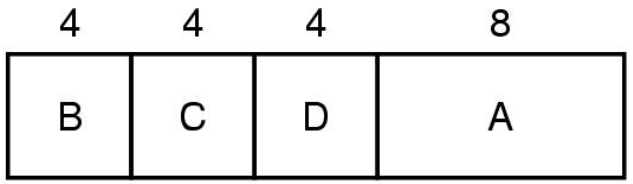
Introduzione al problema dello Scheduling (8)

- Scheduling in sistemi *batch*
  - SJF (shortest job first)
- Scheduling in sistemi *interattivi*
  - Round Robin
  - Code Multiple

Scheduling nei sistemi Batch (1)



(a)



(b)

- Un esempio di scheduling secondo la strategia che privilegia il job più corto (SJF "Shortest Job First")
  - l'insieme dei job da schedulare è noto all'inizio
  - si conosce il tempo di esecuzione *T* di ogni job
  - i job sono schedulati in ordine di *T* crescente
  - SJF minimizza il tempo di turnaround medio
  - non c'è prerilascio

Scheduling nei sistemi Batch (2)

Perché SJF funziona?

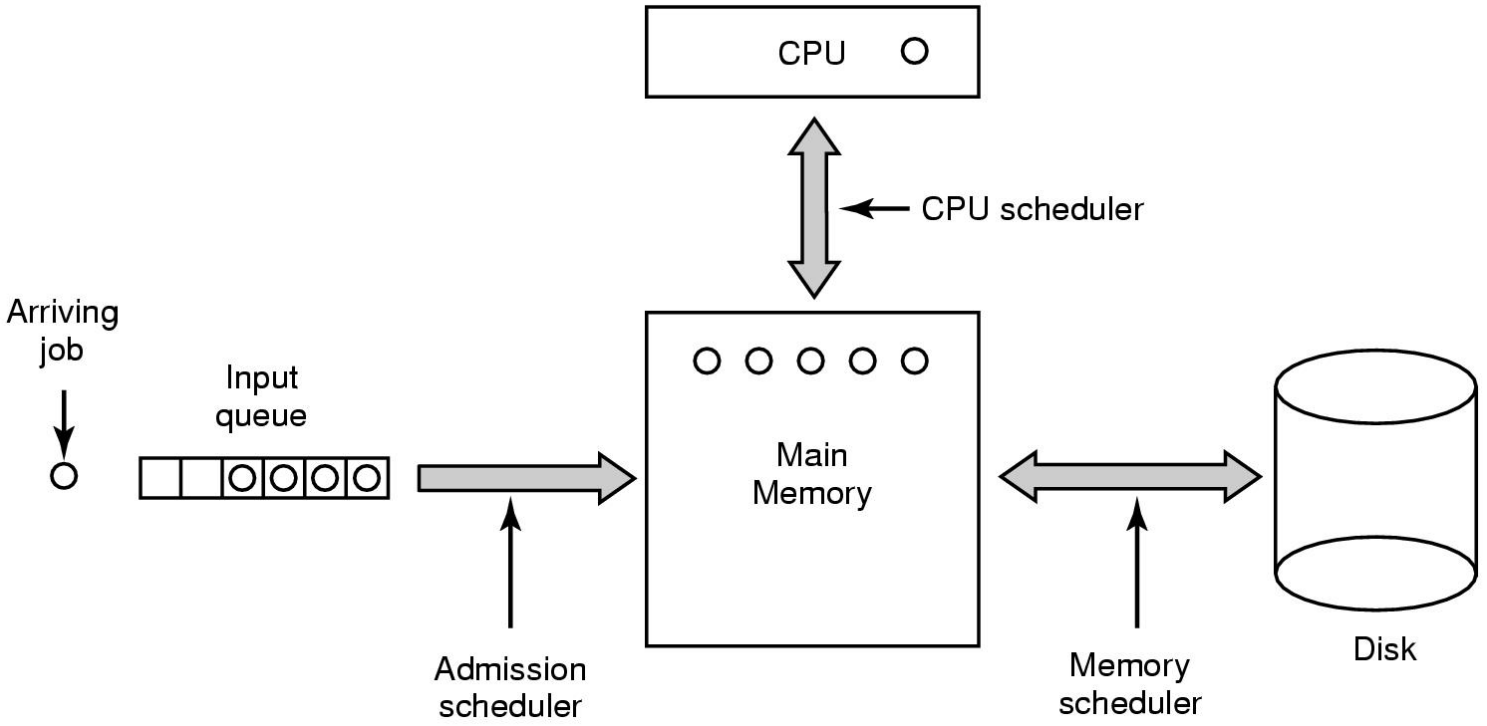
4 job A,B,C,D con tempi di esecuzione a, b, c, d

- turnaround(A) -- a
- turnaround(B) -- $a + b$
- turnaround(C) -- $a + b + c$
- turnaround(D) -- $a + b + c + d$

turnaround totale $4a + 3b + 2c + 1d$

minimo quando a,b,c,d sono in ordine crescente

Scheduling nei sistemi Batch (3)



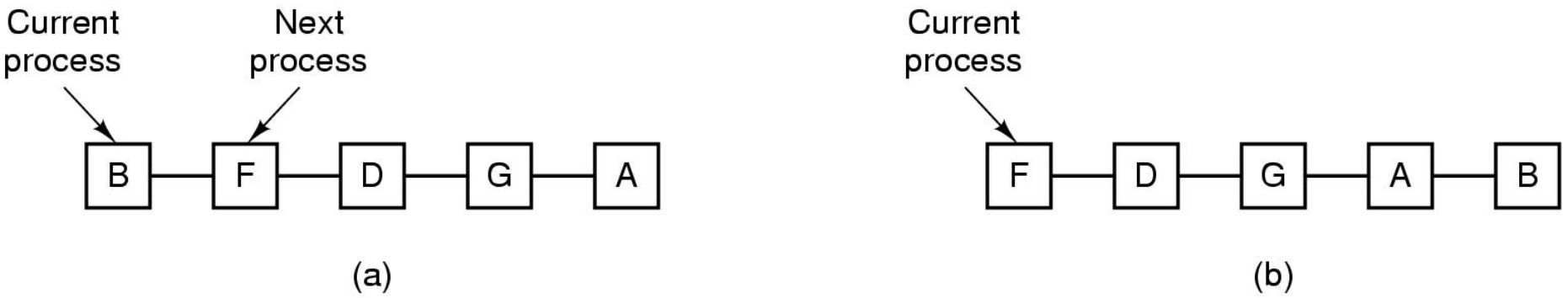
Tre livelli di scheduling

Scheduling nei sistemi Batch (4)

- Admission scheduler
  - decide quali job (sottomessi, memorizzati su disco) ammettere nel sistema (viene creato il processo corrispondente)
- Memory scheduler
  - i job ammessi devono essere caricati in memoria centrale prima di poter essere eseguiti
  - se non tutti i job entrano in MC, il memory scheduler sceglie quali job caricare in memoria e quali tenere su disco (*swapped out*)
- CPU scheduler
  - lo scheduler che abbiamo trattato finora

Scheduling nei sistemi Interattivi

Scheduling *Round Robin* (1)



- (a) lista dei processi pronti
- (b) lista dei pronti dopo che B ha usato il suo *quanto* (*quantum*) di tempo

Scheduling *Round Robin* (2)

- Come fissare il quanto di tempo
  - deve essere abbastanza lungo da ammortizzare il costo di un *context switch* (ordine 1 ms)
  - deve essere abbastanza breve da permettere una risposta veloce agli utenti interattivi
  - in sistemi reali tipicamente 20-120 ms
- RR *non* favorisce i processi I/O bound

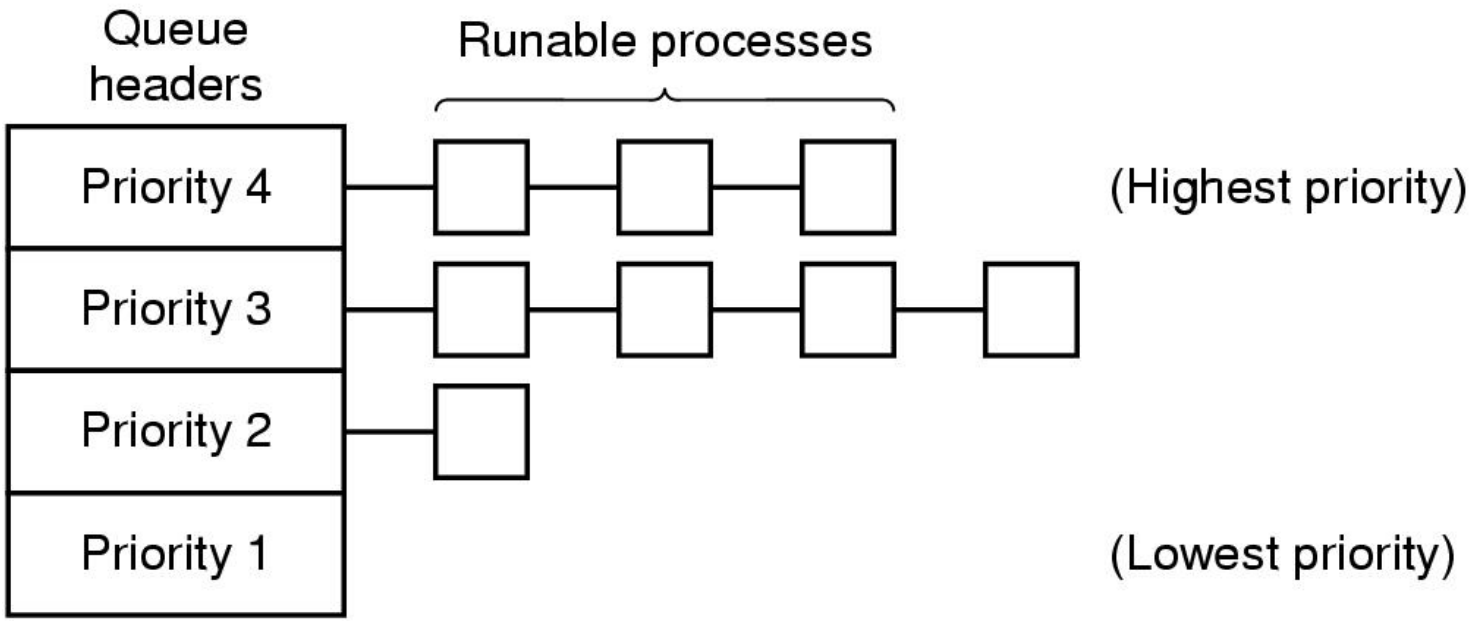
Scheduling con priorità (1)

- Ogni processo ha una priorità
- Ogni volta va in esecuzione il processo a priorità più elevata
- Punti chiave :
  - come assegnare le priorità (statiche, dinamiche…)
  - come evitare attesa indefinita della CPU nei processi a priorità più bassa
  - come individuare i processi I/O bound
    - per elevare la loro priorità

Scheduling con priorità (2)

- Molte strategie per il calcolo della priorità
- Tipicamente :
  - priorità dinamica (es. più elevata per i processi che passano da bloccato a pronto)
  - legata alla percentuale *f* del quanto di tempo che è stato consumato l'ultima volta che il processo è andato in esecuzione (es. proporzionale a $1/f$, favorisce i processi I/O bound)
  - decrescente nel tempo per i processi che rimangono pronti (es. per impedire l'attesa indefinita)

Scheduling con Code multiple (1)



Esempio di algoritmo di scheduling a code multiple con 4 classi di priorità

Scheduling con Code multiple (2)

- Scheduling Round Robin all'interno della classe con priorità più elevata
- I processi che usano tutto il quanto di tempo più di un certo numero di volte vengono passati alla classe inferiore
- Alcuni sistemi danno quanti più lunghi ai processi nelle classi basse (*compute-bound*) per minimizzare l'overhead del cambio di contesto

Scheduling dei Thread (1)

- Lo scheduling dei thread
  - utilizza algoritmi simili a quelli visti finora
  - viene implementato in modo diverso nel thread a livello utente e a livello kernel

Scheduling dei Thread (2)

- Lo scheduling dei thread user level
  - il SO non conosce l'esistenza dei thread, quindi schedula i processi
  - durante l'esecuzione di un processo lo schedulatore della libreria dei thread decide quale thread mandare in esecuzione
  - le interruzioni del clock non sono visibili allo schedulatore di livello utente
  - lo schedulatore può intervenire solo se invocato esplicitamente (es. thread_yield)
  - non c'è prerilascio (all'interno di un singolo processo)

Scheduling dei Thread (3)

- Lo scheduling dei thread kernel level
  - il SO schedula i thread (non i processi)
  - quando un thread si blocca il SO può decidere di mandare in esecuzione un altro thread di quel processo o un thread di un processo diverso
    - può scegiere se pagare il cambio di contesto o no
  - le interruzioni del clock permettono allo schedulatore di tornare in esecuzione alla fine del quento di tempo
    - i quanti di tempo sono assegnati direttamente ai thread
    - si può effettuare prerilascio