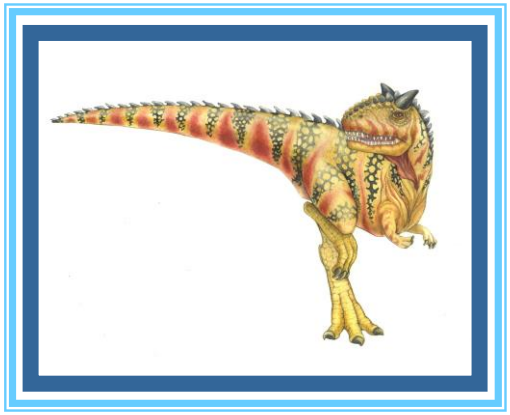
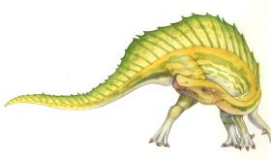
Scheduling della CPU



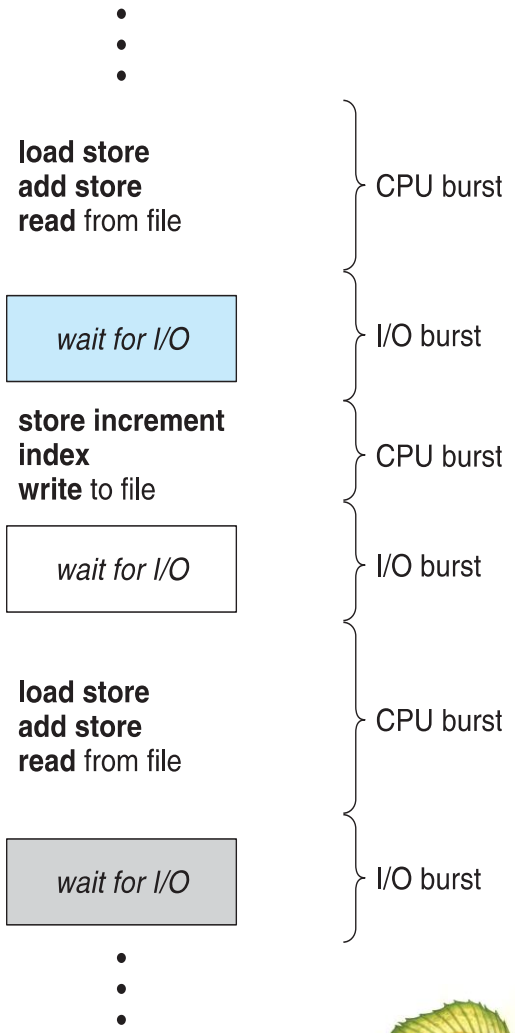
Concetti fondamentali – 1

- Lo scheduling è una funzione fondamentale dei sistemi operativi
  - Si sottopongono a scheduling quasi tutte le risorse di un calcolatore
- Lo scheduling della CPU è alla base dei sistemi operativi multiprogrammati
  - Attraverso la commutazione del controllo della CPU tra i vari processi, il SO rende più "produttivo" il sistema di calcolo



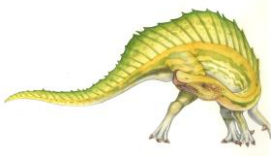
Concetti fondamentali – 2

- Il massimo impiego della CPU è pertanto ottenuto con la multiprogrammazione
- Ciclo di CPU–I/O burst – L'elaborazione di un processo consiste di *cicli* di esecuzione nella CPU ed attese/ utilizzo ai/dei dispositivi di I/O
- I CPU–burst si susseguono agli I/O–burst durante tutta la vita del processo



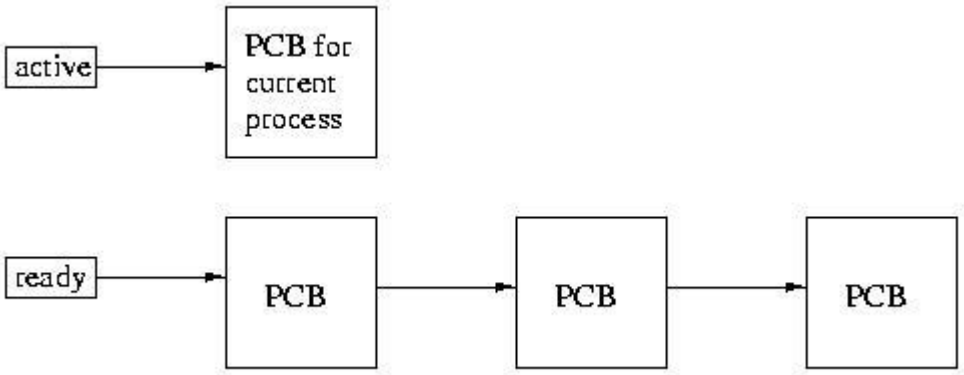
Concetti fondamentali – 3

- Un programma con prevalenza di I/O si definisce *I/O–bound* e produce generalmente molte sequenze di operazioni della CPU di breve durata
- Un programma con prevalenza di elaborazione si definisce *CPU–bound* e produce (poche) sequenze di esecuzione molto lunghe
- Avere contemporaneamente presenti in memoria processi di entrambi i tipi garantisce l'uso intensivo e "bilanciato" di tutte le risorse del sistema



Lo scheduler della CPU – 1

- Lo scheduler della CPU gestisce la coda dei processi pronti, selezionando il prossimo processo cui verrà allocata la CPU
  - Gli elementi nella ready queue sono i PCB dei processi pronti
  - La ready queue può essere realizzata come una coda FIFO (*first–in–first–out*), una coda con priorità, una lista concatenata o un albero
  - La realizzazione (meccanismo) è spesso indipendente dalla politica implementata dallo scheduler

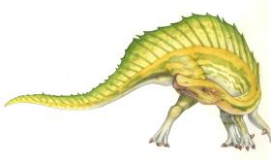


Lo scheduler della CPU – 2

- Lo scheduler della CPU deve prendere una decisione quando un processo:
  1. passa da stato *running* a stato *wait* (es.: richiesta di I/O o attesa terminazione di un processo figlio)
  2. passa da stato *running* a stato *ready* (interrupt)
  3. passa da stato *wait* a stato *ready* (es.: completamento di un I/O)
  4. termina
- Se lo scheduling viene effettuato solo nei casi 1 e 4, si dice che lo schema di scheduling è *nonpreemptive* (senza diritto di prelazione) o *cooperativo*
- Altrimenti si ha uno schema *preemptive*

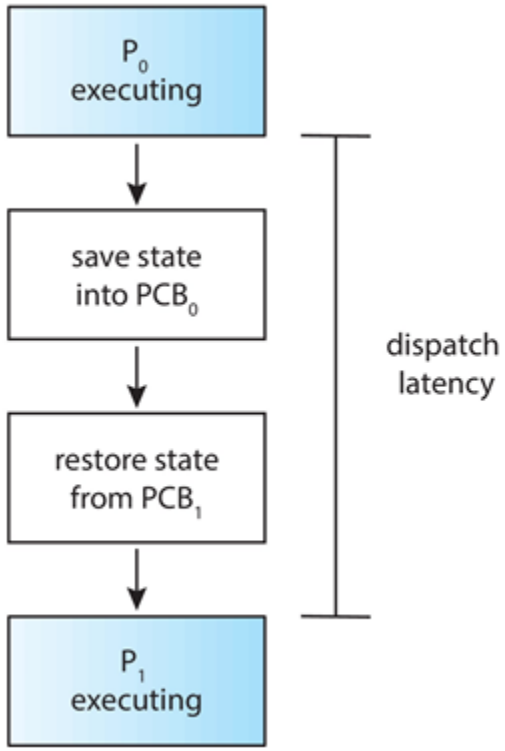
Lo scheduler della CPU – 3

- In caso di scheduler preemptive, sono aspetti critici da tenere in considerazione…
  - L'accesso a dati condivisi (prelazione durante la modifica dei dati con risultati impredicibili)
  - La possibilità di prelazione di processi del kernel (che normalmente modificano dati vitali per il sistema)
  - L'interruzione di operazioni cruciali effettuate dal sistema operativo



Il dispatcher

- Il modulo *dispatcher* passa il controllo della CPU al processo selezionato dallo scheduler; il dispatcher effettua:
  - Context switch
  - Passaggio a modo utente
  - Salto alla posizione corretta del programma utente per riavviarne l'esecuzione
- Latenza di dispatch — è il tempo impiegato dal dispatcher per sospendere un processo e avviare una nuova esecuzione



Context–switch

- Con quale frequenza avvengono i context–switch?
- In Linux, il comando **`vmstat`** fornisce il numero di cambi di contesto

```
$ vmstat 1 3
---cpu---
243
225
339
```

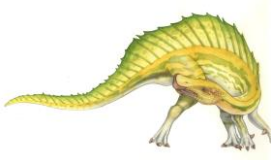
3 righe di output con numero medio di context-switch dall'avvio del sistema e nei due precedenti intervalli di tempo di 1 sec

243 → frequenza media (al secondo)

225, 339 → frequenze durante i due secondi precedenti

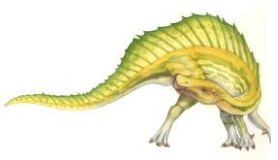
- È anche possibile usare il file system temporaneo **`/proc`**
- Esempio

```
$ cat /proc/2166/status
voluntary_ctxt_switches     150
nonvoluntary_ctxt_switches  8
```



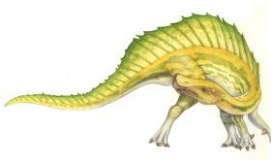
Criteri di scheduling

- Utilizzo della CPU — la CPU deve essere più attiva possibile (comando `top` in Linux, Mac OS e UNIX)
- Throughput (produttività) — numero dei processi che completano la loro esecuzione nell'unità di tempo
- Tempo di turnaround (tempo di completamento) — tempo impiegato per l'esecuzione di un determinato processo
  - Tempo di attesa nella ready queue e nelle code dei dispositivi, tempo effettivo di utilizzo della CPU e dei dispositivi di I/O
- Tempo di attesa — tempo passato dal processo in attesa nella ready queue
- Tempo di risposta — tempo che intercorre tra la sottomissione di un processo e la prima risposta prodotta
  - Nei sistemi time–sharing, il tempo di turnaround può essere influenzato dalla velocità del dispositivo di output



Criteri di ottimizzazione – 1

- Massimo utilizzo della CPU
- Massimo throughput
- Minimo tempo di turnaround
- Minimo tempo di attesa
- Minimo tempo di risposta



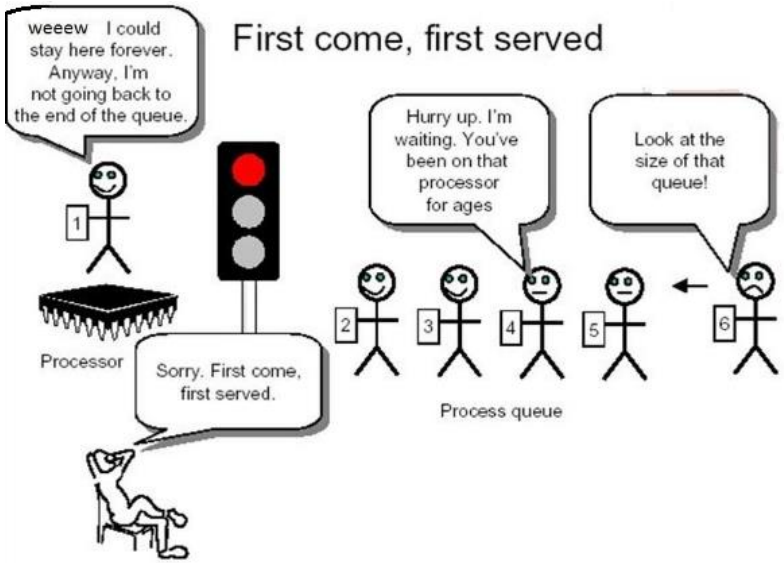
Criteri di ottimizzazione – 2

- Normalmente si ottimizzano i valori medi, ma talvolta è più opportuno ottimizzare i valori minimi/massimi
  - Per esempio, in sistemi interattivi (desktop), per garantire che tutti gli utenti ottengano un buon servizio

    ⇨ ridurre il tempo massimo di risposta
- Per i sistemi time–sharing è invece più significativo ridurre la varianza rispetto al tempo medio di risposta: un sistema con tempo di risposta prevedibile è migliore di un sistema mediamente più rapido, ma molto variabile

Scheduling First–Come–First–Served (FCFS)

- La CPU viene assegnata al processo che la richiede per primo: la realizzazione del criterio FCFS si basa sull'implementazione della ready queue per mezzo di una coda FIFO
- Quando un processo entra nella ready queue, si collega il suo PCB all'ultimo elemento della coda; quando la CPU è libera, viene assegnata al processo che si trova alla testa della ready queue, rimuovendolo da essa

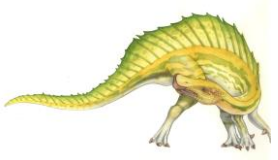


Scheduling FCFS (cont.)

- Esempio 1

Processo	Tempo di burst (millisecondi)
P_1	24
P_2	3
P_3	3

  - I processi arrivano al sistema nell'ordine: P_1, P_2, P_3
  - Per descrivere come si realizza lo scheduling si usa un diagramma di Gantt
    - Un istogramma che illustra una data pianificazione, includendo i tempi di inizio e di fine di ogni processo



Scheduling FCFS (cont.)

Processo	Tempo di burst (millisecondi)
P_1	24
P_2	3
P_3	3

- Il diagramma di Gantt per lo scheduling FCFS è:

- Tempi di attesa in msec: $P_1 \rightarrow 0$, $P_2 \rightarrow 24$, $P_3 \rightarrow 27$
- Tempo medio di attesa $T_a = (0+24+27)/3 = 17$msec

Scheduling FCFS (cont.)

- Supponiamo che l'ordine di arrivo dei processi sia

  P_2 , P_3 , P_1

- Il diagramma di Gantt diventa:

Processo	Tempo di burst (millisecondi)
P_1	24
P_2	3
P_3	3

- Tempi si attesa in msec: $P_1 \rightarrow 6$, $P_2 \rightarrow 0$, $P_3 \rightarrow 3$
- Tempo medio di attesa $T_a = (6+0+3)/3 = 3$msec
- Molto inferiore al precedente: in questo caso, non si verifica l'*effetto convoglio*, per cui processi di breve durata devono attendere che un processo lungo liberi la CPU
  - Situazione in cui vi è un solo processo CPU–bound e molti processi I/O–bound

Scheduling Shortest–Job–First (SJF)

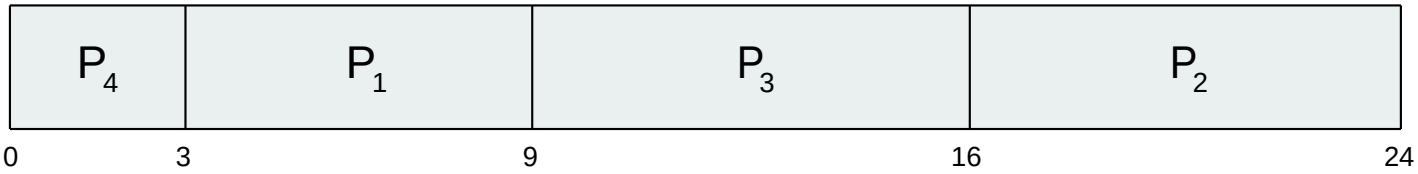
- Si associa a ciascun processo la lunghezza del suo burst di CPU successivo; si opera lo scheduling in base alla brevità dei CPU–burst
- Due schemi:
  - non–preemptive — dopo che la CPU è stata allocata al processo, non gli può essere prelazionata fino al termine del CPU burst corrente
  - preemptive — se arriva un nuovo processo con burst di CPU minore del tempo rimasto per il processo corrente, il nuovo processo prelaziona la CPU: SRTF, *Shortest–Remaining–Time–First*
- SJF è *ottimo* — minimizza il tempo medio di attesa
  - Difficoltà nel reperire l'informazione richiesta

Scheduling SJF non–preemptive

- Esempio 2

Processo	Tempo di burst
P_1	6
P_2	8
P_3	7
P_4	3

  - Diagramma di Gantt



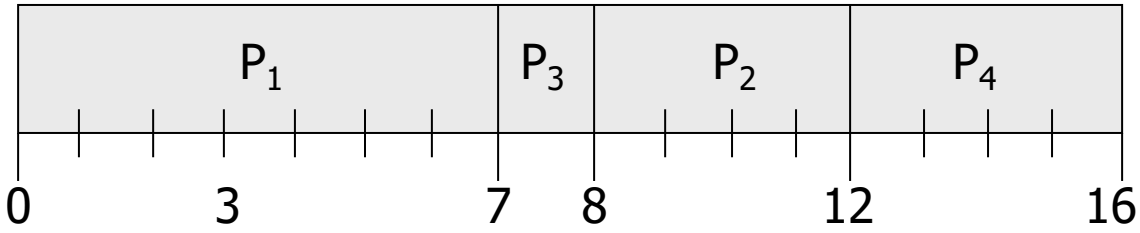
  - Tempo medio di attesa T_a=(3+16+9+0)/4=7msec
  - Usando lo scheduling FCFS, T_a=10.25msec

Scheduling SJF non–preemptive (cont.)

- Esempio 3

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

  - Diagramma di Gantt



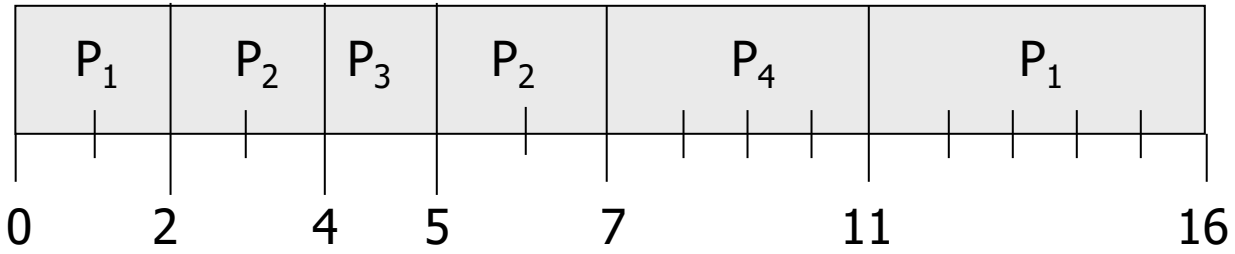
  - Tempo medio di attesa T_a=(0+6+3+7)/4=4msec

Scheduling SJF preemptive (SRTF)

- Esempio 4

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

  - Diagramma di Gantt



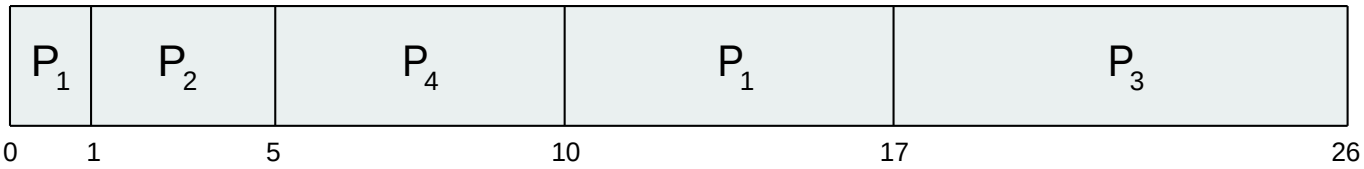
  - Tempo medio di attesa $T_a = (9+1+0+2)/4 = 3$msec

Scheduling SJF preemptive (SRTF)

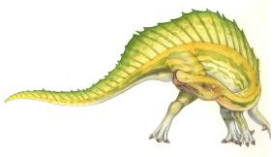
- Esempio 5

Processo	Tempo di arrivo	Tempo di burst
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- Diagramma di Gantt



- Tempo medio di attesa
- T_a=[(10–1)+(1–1)+(17–2)+(5–3)]/4=26/4=6.5msec
- Usando SJF non preemptive, T_a=7.75msec



Scheduling a priorità

- Un valore di priorità (intero) viene associato a ciascun processo
- La CPU viene allocata al processo con la priorità più alta (spesso, intero più basso≡priorità più alta)
  - Preemptive
  - Non–preemptive
- SJF è uno scheduling a priorità in cui la priorità è calcolata in base al successivo tempo di burst
- Problema: *Starvation* ("inedia", blocco indefinito) – i processi a bassa priorità potrebbero non venir mai eseguiti
- Soluzione: *Aging* (invecchiamento) – aumento graduale della priorità dei processi che si trovano in attesa nel sistema da lungo tempo

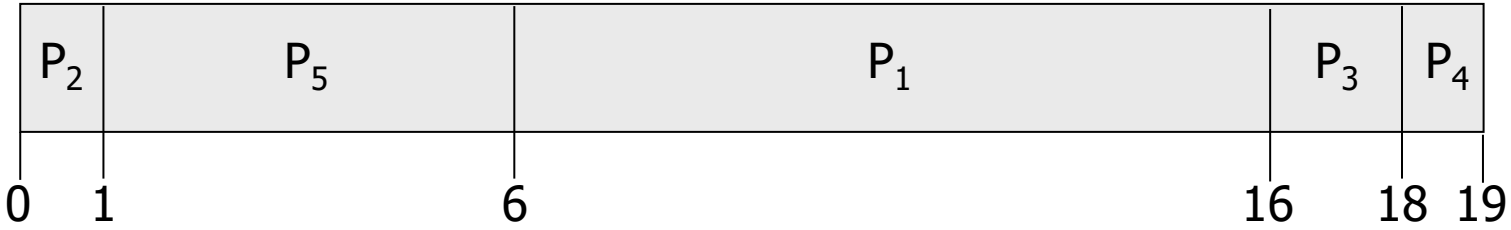
Scheduling a priorità (cont.)

- Esempio 6

Processo	Tempo di burst	Priorità
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Priorità massima

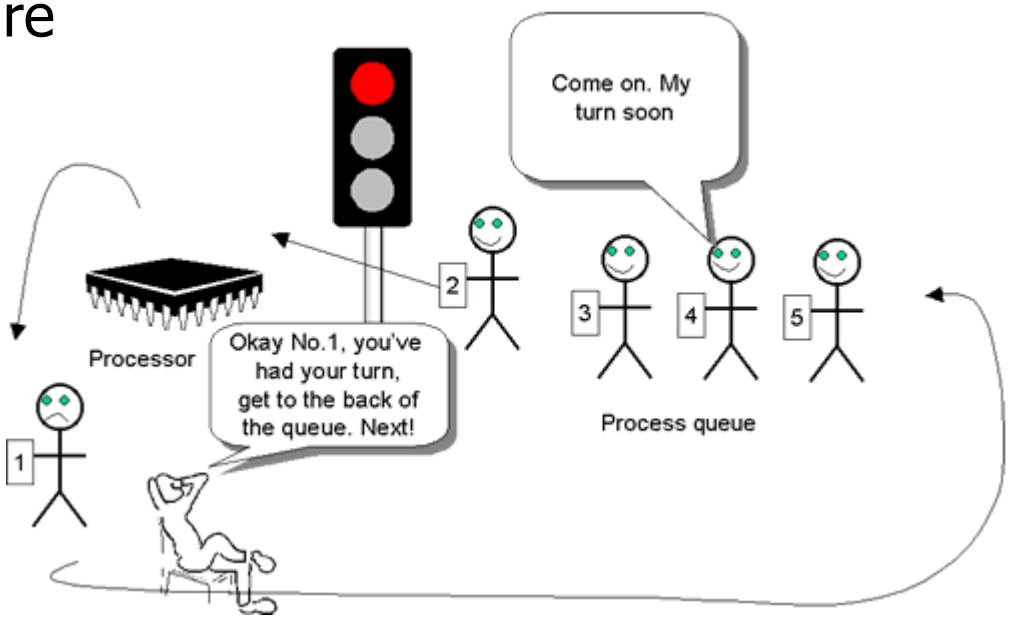
- Il diagramma di Gantt è:



- Tempi di attesa: $P_1 \rightarrow 6$, $P_2 \rightarrow 0$, $P_3 \rightarrow 16$, $P_4 \rightarrow 18$, $P_5 \rightarrow 1$
- Tempo medio di attesa $T_a = (6+0+16+18+1)/5 = 8.2$msec

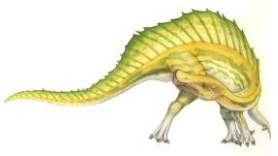
Scheduling Round Robin (RR)

- A ciascun processo viene allocata una piccola quantità di tempo di CPU, un *quanto di tempo* (*time slice*), generalmente 10–100 millisecondi
  - Dopo un quanto di tempo, il processo è forzato a rilasciare la CPU e viene accodato alla ready queue
  - La ready queue è trattata come una coda (FIFO) circolare



Scheduling RR (cont.)

- Se vi sono n processi nella ready queue ed il quanto di tempo è q, ciascun processo occupa $1/n$ del tempo di CPU in frazioni di, al più, q unità di tempo; nessun processo attende per più di $(n-1)\times q$ unità di tempo
- Il timer interrompe la CPU allo scadere del quanto per programmare il prossimo processo
- Prestazioni:
  - q grande ⇨ FCFS
  - q piccolo ⇨ q deve essere grande rispetto al tempo di context switch (che, tuttavia, è normalmente $<10\mu$sec), altrimenti l'overhead è troppo alto

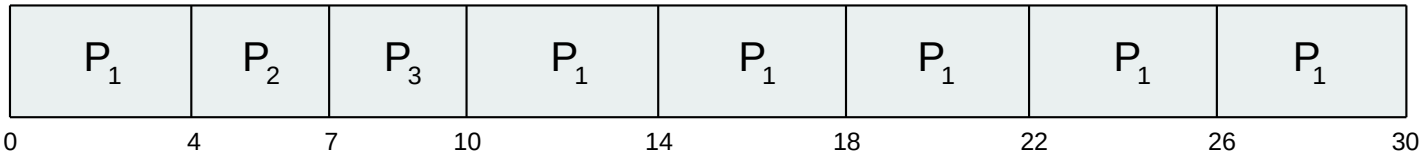


Scheduling RR (cont.)

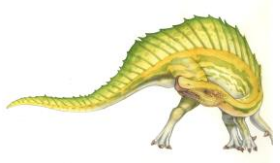
- Esempio 7

Processo	Tempo di burst
P_1	24
P_2	3
P_3	3

- Il diagramma di Gantt con $q = 4$ è:



- In genere si ha un tempo medio di attesa e di turnaround maggiore rispetto a SJF, tuttavia si ottiene un miglior tempo medio di risposta
- Tempi di attesa: $P_1 \rightarrow 6$, $P_2 \rightarrow 4$, $P_3 \rightarrow 7$
- Tempo medio di attesa $T_a = (6+4+7)/3 = 5.\overline{6}$msec

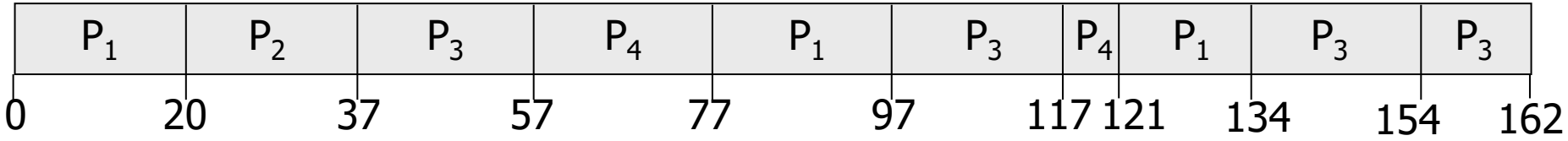


Scheduling RR (cont.)

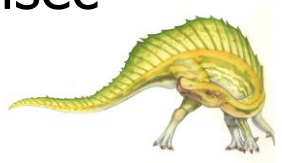
- Esempio 8

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

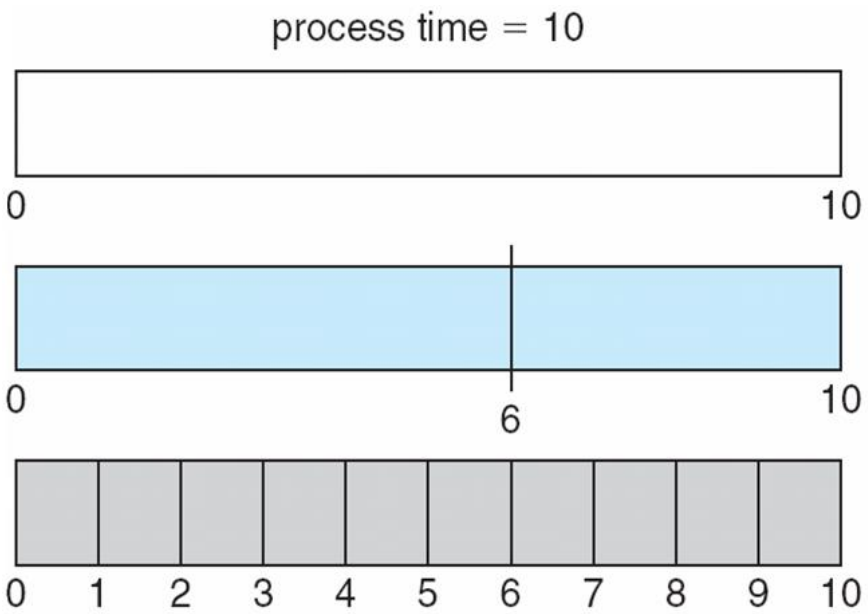
- Il diagramma di Gantt con $q = 20$ è:



- Tempi di attesa: $P_1 \rightarrow 81$, $P_2 \rightarrow 20$, $P_3 \rightarrow 94$, $P_4 \rightarrow 97$
- Tempo medio di attesa $T_a = (81+20+94+97)/4 = 73$msec



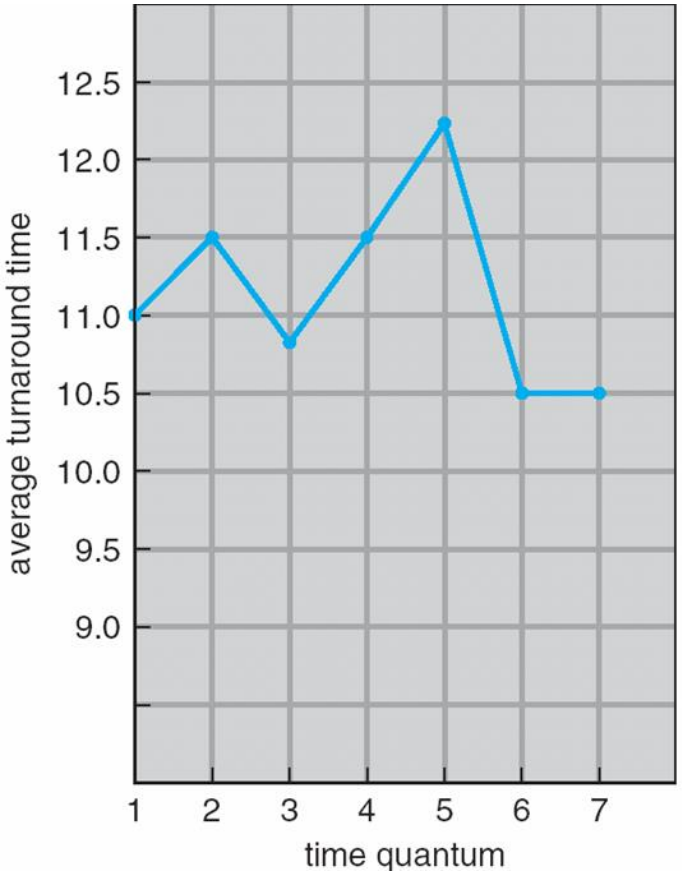
Quanto di tempo e tempo di context–switch



quantum	context switches
12	0
6	1
1	9

Un quanto di tempo minore incrementa il numero di context switch

Quanto di tempo e tempo di turnaround



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Empiricamente: il quanto di tempo deve essere più lungo dell'80% dei CPU burst

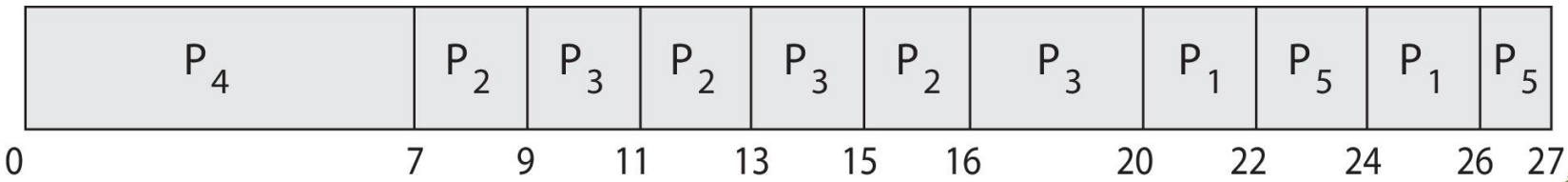
Variazione del tempo medio di turnaround in funzione della lunghezza del quanto di tempo

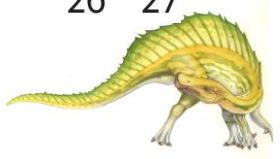
Scheduling a priorità con RR

- Esempio 9

Processo	Tempo di burst	Priorità
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

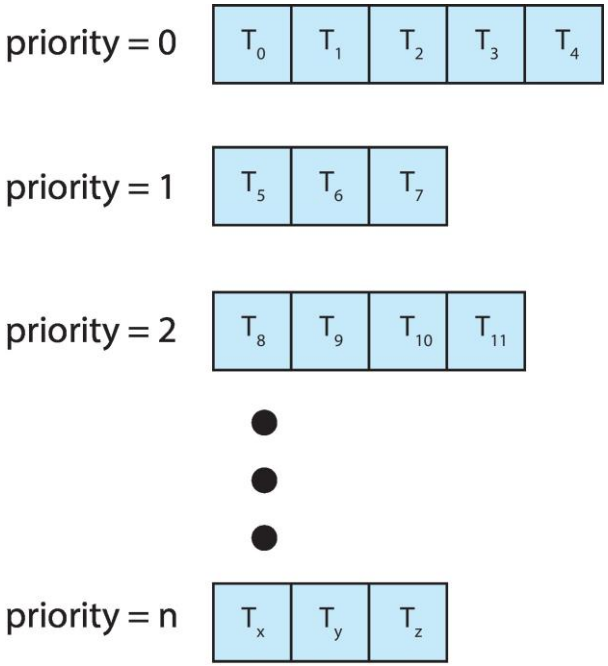
- Viene eseguito il processo a priorità maggiore; a parità di priorità, si usa RR
- Il diagramma di Gantt con $q = 2$ è:





Scheduling con code multiple – 1

- Nello scheduling a priorità tutti i processi possono essere collocati in una singola coda, da cui lo scheduler seleziona il prossimo processo da eseguire
  - La ricerca può costare $\mathcal{O}(n)$
- Nella pratica, risulta più conveniente organizzare i processi in code distinte
  - Lo scheduler assegna la CPU al processo "in testa" alla coda a priorità più alta
- L'approccio funziona bene anche per lo scheduling combinato con RR
  - Se ci sono più processi in una coda con una data priorità, questi verranno eseguiti con scheduling circolare

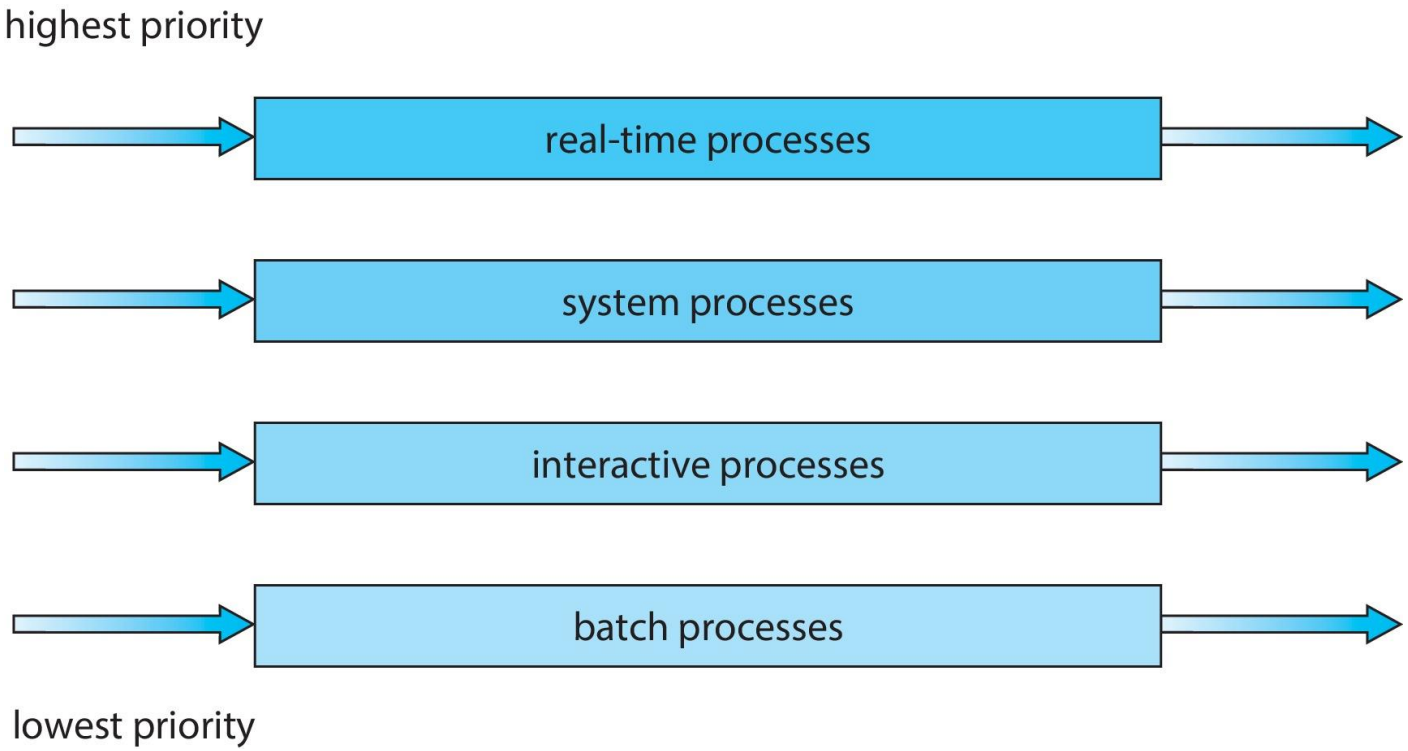


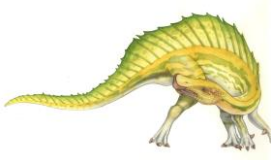
Scheduling con code multiple – 2

- Esempio 10
  - La ready queue è suddivisa nelle due code
    - foreground (interattiva)
    - background (batch)
  - Ogni coda ha il suo proprio algoritmo di scheduling
    - foreground: RR
    - background: FCFS
  - È necessario effettuare lo scheduling tra le code
    - *Scheduling a priorità fissa*: si servono tutti i processi foreground poi quelli background
      ⇨ Rischio di starvation
    - *Time slice*: ciascuna coda occupa un certo tempo di CPU che suddivide fra i propri processi; ad esempio…
      - 80% per foreground con RR
      - 20% per background con FCFS

Scheduling con code multiple – 3

- I processi si assegnano in modo permanente ad una coda, generalmente secondo qualche caratteristica (invariante) del processo



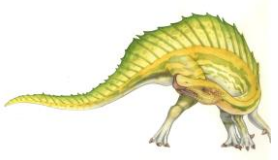


Code multiple con feedback – 1

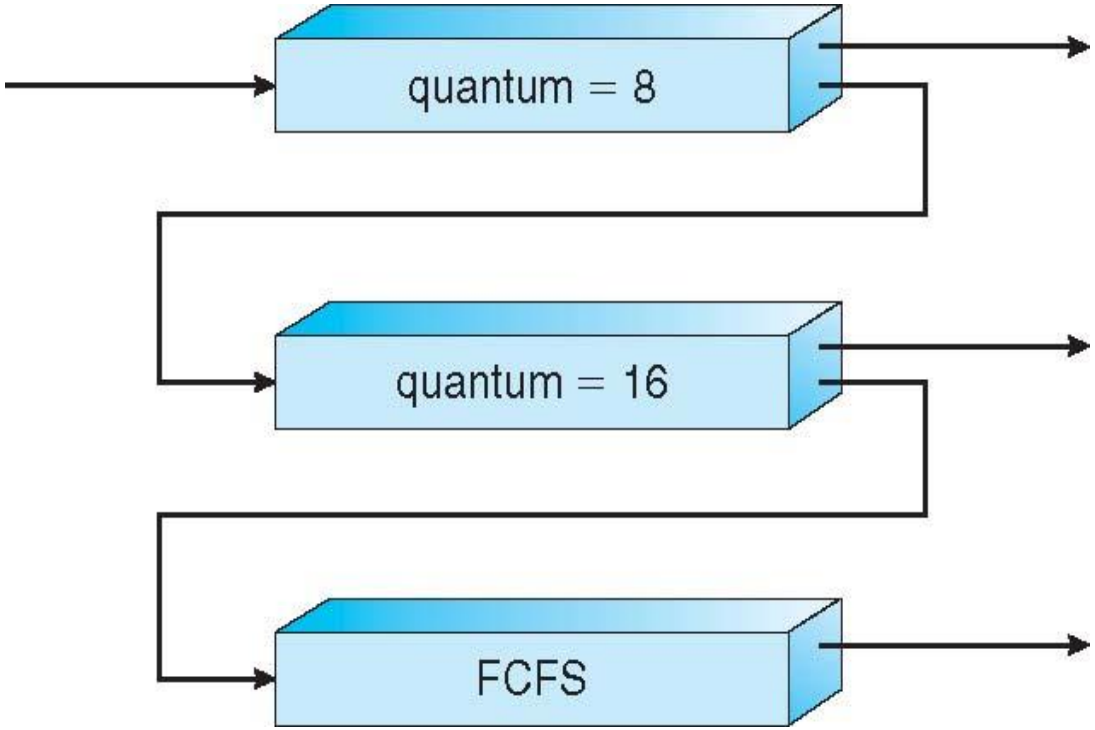
- Un processo può spostarsi fra le varie code; si può implementare l'aging
- Lo scheduler a code multiple con feedback è definito dai seguenti parametri:
  - Numero di code
  - Algoritmo di scheduling per ciascuna coda
  - Metodo impiegato per determinare quando spostare un processo in una coda a priorità maggiore
  - Metodo impiegato per determinare quando spostare un processo in una coda a priorità minore
  - Metodo impiegato per determinare in quale coda deve essere posto un processo quando entra nel sistema

Code multiple con feedback – 2

- Esempio 11
  - Tre code:
    - Q_0 – RR con quanto di tempo di 8 millisecondi
    - Q_1 – RR con quanto di tempo di 16 millisecondi
    - Q_2 – FCFS
- Scheduling
  - Un nuovo processo viene immesso nella coda Q_0, che è servita con RR; quando prende possesso della CPU il job riceve 8 millisecondi; se non termina, viene spostato nella coda Q_1
  - Nella coda Q_1 il job è ancora servito RR e riceve ulteriori 16 millisecondi; se ancora non ha terminato, viene spostato nella coda Q_2, dove verrà servito con criterio FCFS all'interno dei cicli di CPU lasciati liberi dai processi delle code Q_0 e Q_1



Code multiple con feedback – 3



- L'idea sottesa all'algoritmo di scheduling è quella di separare i processi in base alle loro caratteristiche d'uso della CPU
  - ⇨ Massima priorità con CPU–burst brevi

Esempio 1

- Esercizio

Un insieme di task indipendenti, A, B, C e D, devono essere eseguiti su di un unico processore. I task possono venire elaborati più di una volta. Appena un task ha terminato la propria esecuzione, è pronto per la successiva (che non necessariamente durerà lo stesso tempo). I tempi di arrivo e di esecuzione sono descritti nella seguente tabella:

Task	Tempo di arrivo	1° esecuzione	2° esecuzione	3° esecuzione
A	0	10	6	4
B	2	3	2	2
C	3	2	–	–
D	5	1	1	–

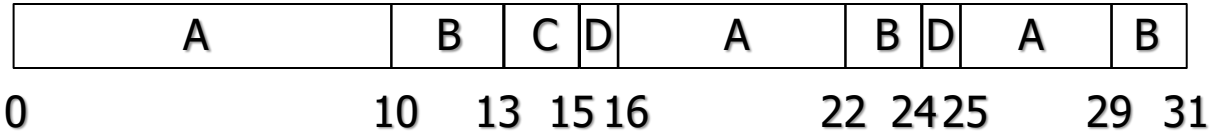
Si tracci il diagramma di Gantt per lo scheduling FCFS e si calcoli il tempo medio di attesa. Lo scheduling generato soffre dell'*effetto convoglio*? Motivare la risposta data.

Esempio 1

- Soluzione

Task	Tempo di arrivo	1° esecuzione	2° esecuzione	3° esecuzione
A	0	10	6	4
B	2	3	2	2
C	3	2	–	–
D	5	1	1	–

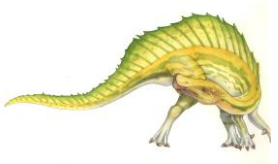
Diagramma di Gantt



$T_a(A)=9$, $T_a(B)=22$, $T_a(C)=10$, $T_a(D)=18$

$T_a = (9+22+10+18)/4 = 14.75$ msec

Si verifica l'effetto convoglio perché il processo A, che viene sempre eseguito per primo, ha burst significativamente più lunghi di tutti gli altri processi.



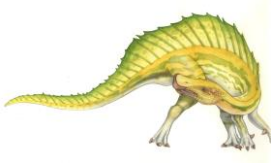
Esempio 2

Esercizio

Si considerino i processi P_1, P_2 e P_3. Ciascun processo esegue un CPU–burst ed un I/O–burst, quindi nuovamente un CPU–burst ed un I/O–burst ed infine un ultimo CPU–burst. La lunghezza dei burst ed il tempo di arrivo dei processi (in millisecondi) è riportato in tabella:

Processo	Burst1	I/O_1	Burst2	I/O_2	Burst3	Arrivo
P_1	2	4	2	2	2	0
P_2	2	2	3	3	1	1
P_3	1	2	1	1	1	1

Si disegnino i diagrammi di Gantt che illustrano l'esecuzione dei tre processi utilizzando FCFS ed RR con quanto di tempo pari a 2. Se il termine di un servizio di I/O ed un timeout della CPU si verificano nello stesso istante, si assegni la precedenza al processo che ha appena terminato il proprio I/O. Si calcoli il tempo medio di attesa ed il tempo medio di turnaround nei due casi.

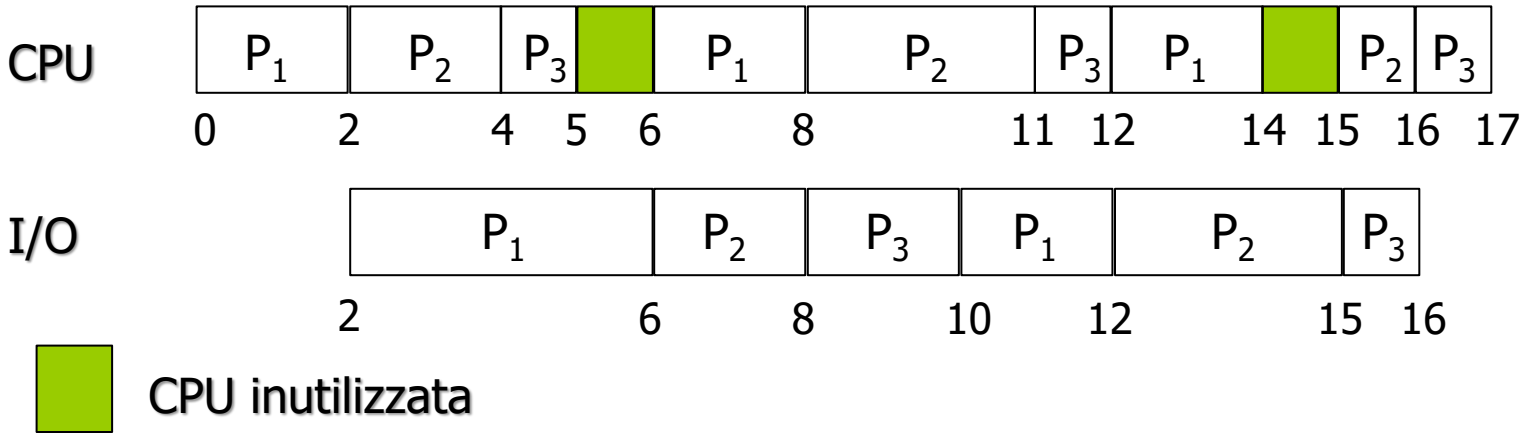


Esempio 2

- Soluzione FCFS

  Diagrammi di Gantt

Processo	Burst1	I/O_1	Burst2	I/O_2	Burst3	Arrivo
P_1	2	4	2	2	2	0
P_2	2	2	3	3	1	1
P_3	1	2	1	1	1	1

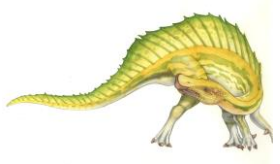


CPU inutilizzata

$T_a = (0+1+4)/3 = 1.\overline{6}\text{msec}$

$T_t = (14+15+16)/3 = 15\text{msec}$

- **Nota:** Nel tempo di attesa deve essere considerato esclusivamente il tempo trascorso dai processi nella ready queue (senza considerare il tempo di attesa/servizio di I/O)

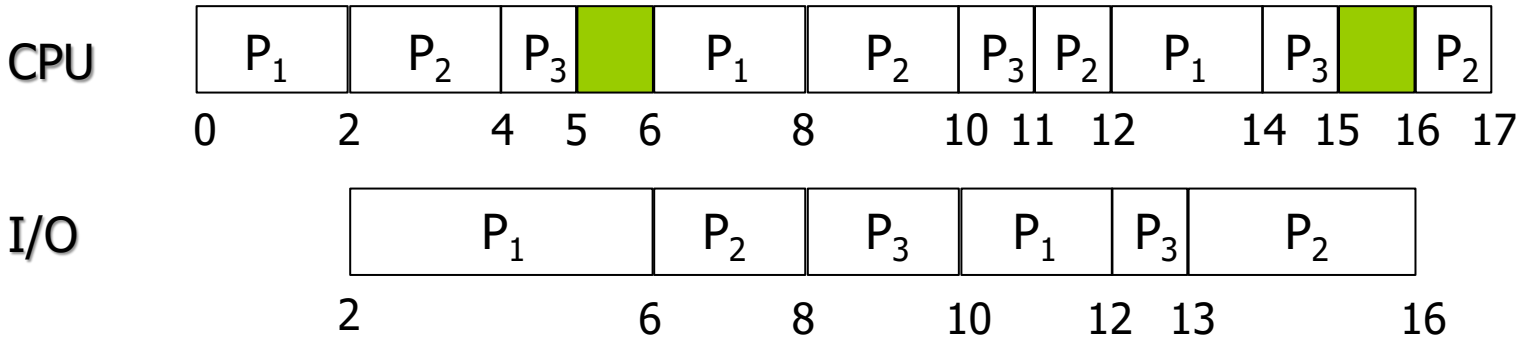


Esempio 2

- Soluzione RR, q=2

  Diagrammi di Gantt

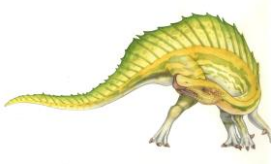
Processo	Burst1	I/O_1	Burst2	I/O_2	Burst3	Arrivo
P_1	2	4	2	2	2	0
P_2	2	2	3	3	1	1
P_3	1	2	1	1	1	1



CPU inutilizzata

$T_a = (0+2+4)/3 = 2\text{msec}$

$T_t = (14+16+14)/3 = 14.\overline{6}\text{msec}$



Esempio 3

- Esercizio

Supponiamo di ricevere 5 job (A,B,C,D,E) tali che:

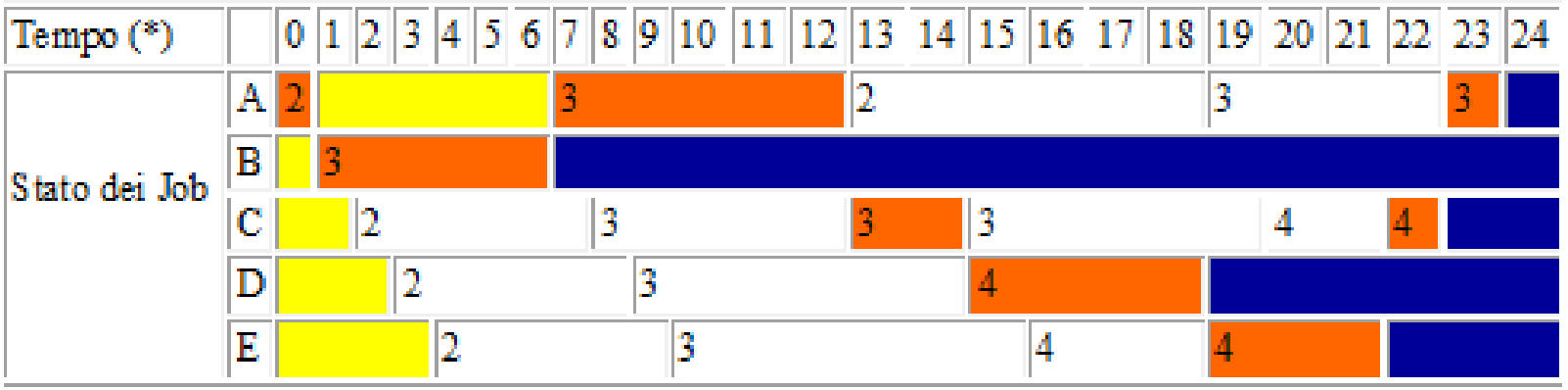
Job	Tempo di arrivo	CPU burst	Priorità
A	0	8	2
B	1	6	3
C	2	3	2
D	3	4	2
E	4	3	2

Descrivere la sequenza di esecuzione dei job (ad es. tramite Gantt chart) e calcolare il tempo totale di completamento ottenuto con uno scheduling a code multiple di priorità e feedback, dove ogni coda è gestita con la strategia FCFS. Il feedback è definito come segue: la priorità diminuisce di 1 (fino al livello base 1) se si passano più di 6 unità di tempo consecutive in esecuzione nella CPU ed aumenta di 1 per ogni 6 unità di tempo passate in attesa in una qualsiasi coda.

Esempio 3

- Soluzione

Job	Tempo di arrivo	CPU burst	Priorità
A	0	8	2
B	1	6	3
C	2	3	2
D	3	4	2
E	4	3	2



- Istante di completamento:

$$A \rightarrow 24,\ B \rightarrow 7,\ C \rightarrow 23,\ D \rightarrow 19,\ E \rightarrow 22$$

- Nota: La preemption avviene al tempo 13 – C sostituisce A – e al 15 – D sostituisce C. Al tempo 19, la coda di priorità 4 contiene solo E, mentre C entra in coda al tempo 20