Lezione 6

SCHEDULING DEI PROCESSI

- Concetti fondamentali
- Criteri di scheduling
- Algoritmi di scheduling
- Il ruolo delle priorita'
- Valutazione degli algoritmi

Concetti fondamentali

Il massimo impiego della CPU è ottenuto con la multiprogrammazione.

⇩

Presenza in memoria di numerosi processi in attesa di essere eseguiti

⇩

Chi decide quale processo deve accedere alla CPU?

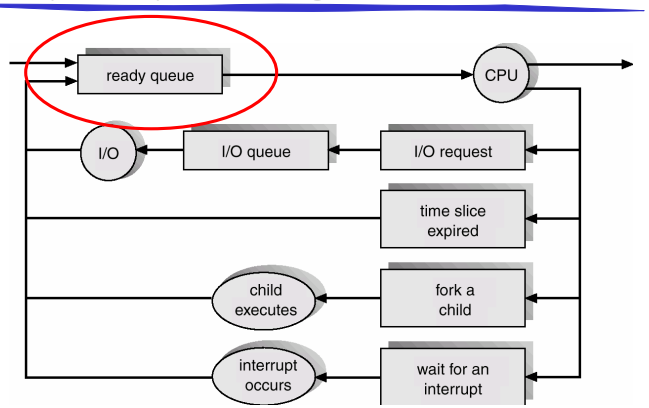
⇩

SCHEDULER

Vari tipi di scheduler

- Scheduler a **lungo termine** (o scheduler dei job):
  - seleziona quali processi possono competere per le risorse del calcolatore
  - Controlla il numero di processi in un sistema (grado di multiprogrammazione)
  - Puo' essere chiamato ogni volta che si crea un nuovo processo (secondi o minuti)
  - Puo' essere assente nei moderni S.O. (UNIX)
- Scheduler a **medio termine** (oppure *swapper*):
  - rimuove processi dalla memoria (e dalla contesa per la CPU)
  - E' responsabile del numero di processi presenti in memoria e facilita il compito dello scheduler a breve termine
  - Assieme allo scheduler a lungo termine determina la composizione delle code dei processi
- Scheduler a **breve termine** (oppure scheduler della CPU):
  - seleziona quale processo debba essere eseguito successivamente, ed alloca la CPU.
  - Deve essere chiamato molto spesso (~ 100 msec)
  - Deve essere veloce (~ 1 msec)
  - E' responsabile dei tempi di attesa

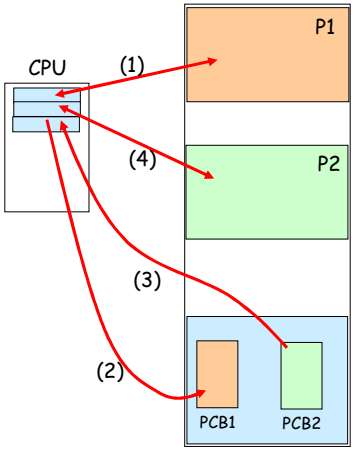
Ci occupiamo soprattutto degli scheduler a breve termine



dispatcher

- Il modulo *dispatcher* passa il controllo della CPU al processo selezionato dallo scheduler a breve termine;
- Il dispatcher effettua il cambio di contesto (context switch)
- *Latenza di dispatch* — è il tempo impiegato dal dispatcher per sospendere un processo e avviare l' esecuzione di un nuovo processo

Context switch



1 la cpu lavora in mod. utente. I registri della CPU contengono informazioni relativi a P1 (program counter, puntatori allo stack, PID, stato,…)

2 la cpu passa in modalita' sistema e salva il contenuto dei registri nel PCB1

3 la cpu carica nei registri il contenuto del PCB2 precedentemente salvato e passa in mod. utente

4 la cpu ha il controllo di P2

Preemptive / non preemptive

Uno scheduler puo' operare in maniera
preemptive (con prelazione) o nonpreemptive (senza prelazione)

- Preemptive
  - I processi possono essere rimossi dalla CPU
  - Puo' portare ad un miglioramento delle prestazioni
  - Essenziale per s.o. interattivi e time-sharing
- Nonpreemptive
  - I processi mantengono l'uso della CPU fino al loro completamento
  - Processi poco importanti possono far ritardare quelli importanti

Preemptive / non preemptive

la politica preemptive e' in genere costosa a causa dei continui cambio di contesto. Va quindi usata solo quando necessaria

- I sistemi operativi batch sono soprattutto non preemptive, in quanto non ci sono utenti in attesa al terminale
- I sistemi operativi time sharing general purpose che favoriscono l'uso interattivo del calcolatore, sono soprattutto preemptive
- I sistemi operativi real time sono spesso non preemptive, per non aumentare l'overhead del sistema e perche' in genere compiono mansioni specifiche e di breve durata e quindi uno scheduler preemptive non e' necessario

Priorita'

Non tutti i processi sono della stessa importanza

Spesso gli scheduler usano delle priorita'
per determinare il processo da eseguire,
dando priorita' maggiore a processi piu' importanti

- Priorita' statica
  - Non cambia durante l'esecuzione del processo
  - facile da realizzare
- Priorita' dinamica
  - cambia durante l'esecuzione del processo
  - difficile da realizzare e con un maggior overhead

Es: priorita' Windows 2000

Processi importanti ⟷ Processi poco importanti

Windows	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

Priorita' variabile

- Una ready-queue per ogni classe di processi
- importanza differente: alta (es. sistema o finestre in foreground) bassa (es. finestre ridotte o in background)
- priorita' variabile: aumenta (es. e' in attesa di eventi) diminuisce (es. se finisce il timeslice)

Obiettivi di scheduling

- **Massimo utilizzo di CPU** — la CPU deve essere più attiva possibile.
- **Massimo Throughput** — numero di processi che completano la loro esecuzione nell'unità di tempo.
- **Minimo Tempo di completamento** — tempo medio di esecuzione di un processo (comprese le attese).
- **Minimo Tempo di attesa** — tempo speso dal processo in attesa nella ready queue.
- **Minimo Tempo di risposta** — tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta.

Obiettivi di scheduling

- alcuni obiettivi sono complementari (ad es. massimo throughput e massimo uso della CPU)
- altri obiettivi sono in opposizione tra loro (ad es. minimo tempo di attesa di un processo e massimo utilizzo delle risorse)

Obiettivi differenti per sistemi differenti

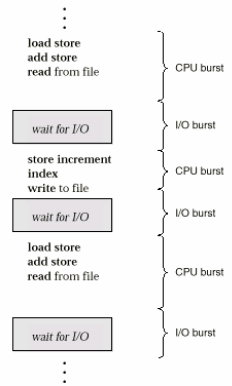
Obiettivi di scheduling

- obiettivi generali
  - equita' (dare ad ogni processo una porzione equa della CPU)
  - controllo (verificare che le politiche vengano messe in atto)
  - bilanciamento (tenere occupate tutte le parti del sistema)
  - uso della CPU (tenere sempre occupata la CPU)
- sistemi batch
  - throughput (massimizzare job per unita' di tempo)
  - turnaround time (minimizzare il tempo medio di esecuzione)
- sistemi interattivi
  - response time (minimizzare i tempi di risposta)
- sistemi real time
  - deadline: rispettare le scadenze

Definizione: Burst di CPU e I/O

- *Ciclo di CPU-I/O burst* — L'esecuzione di un processo consiste di *cicli* di esecuzione da parte della CPU ed attese di I/O.

(burst = raffica)



Distribuzione dei burst di CPU

Un obiettivo comune: uso della CPU

I processi possono essere classificati in:

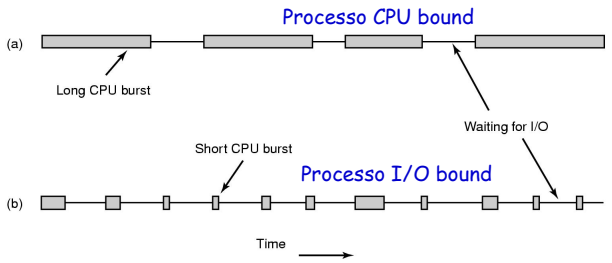
- *Processi I/O-bound*: impiegano più tempo effettuando I/O rispetto al tempo impiegato per elaborazioni
  - in generale, si hanno molti *burst* di CPU di breve durata.
  - Utilizzano le code dei dispositivi
- *Processi CPU-bound*: impiegano più tempo effettuando elaborazioni
  - in generale, si hanno pochi burst di CPU di lunga durata.
  - utilizzano la ready queue

OBIETTIVO

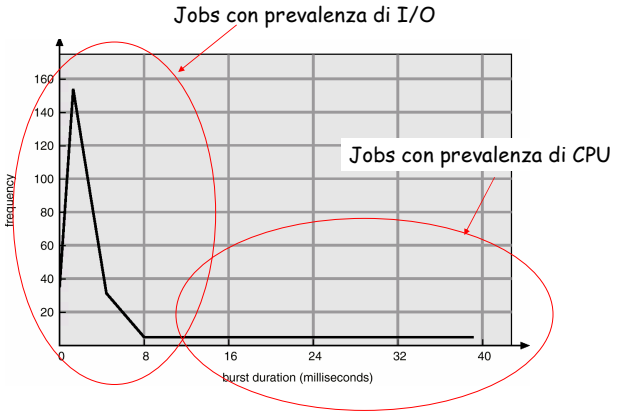
Avere sempre processi presenti nelle code dei processi pronti e dei dispositivi

2 classi di processi

- processi con prevalenza di CPU (CPU bound)
- processi con prevalenza di I/O (I/O bound)



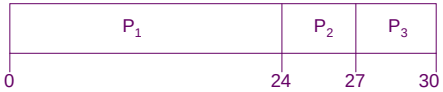
Uso della CPU



Scheduling First-Come-First-Served (FCFS)

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.

Scheduling FCFS

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.

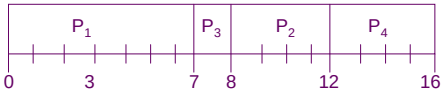
Scheduling Shortest-Job-First (SJF)

- Si associa a ciascun processo la lunghezza del suo burst di CPU successivo. Si opera lo scheduling in base al tempo rimanente dei processi.
- Due schemi:
  - **non-preemptive** — una volta 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. Questo schema è noto come ***Shortest-Remaining-Time-First*** (**SRTF**).
- **SJF** è *ottimale* — rende minimo il tempo medio di attesa per un dato insieme di processi.

Scheduling SJF 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):



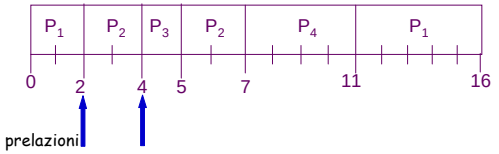
- Tempo medio di attesa = (0 + 6 + 3 + 7)/4 = 4.
- Tempo medio di completamento = (7 + 10 + 4 + 11)/4 = 8.

Scheduling SJF 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.

Scheduling Shortest-Job-First (SJF)

Lo scheduling SJF e' ottimale,
in quanto minimizza i tempi medi di attesa

Tale stima e' calcolata sulla previsione dei tempi di utilizzo della CPU che non sono informazioni presenti nel PCB

Come stimare i tempi di uso della CPU

Predizione della lunghezza del CPU burst successivo

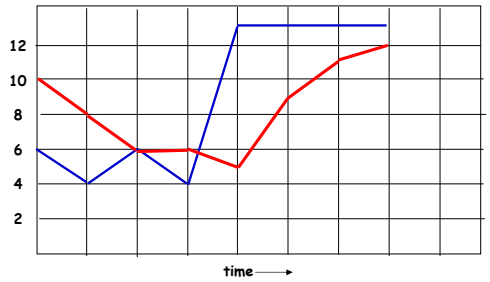
- Può essere stimato utilizzando la lunghezza dei burst di CPU precedenti, impiegando una media esponenziale.

1. t_n = lunghezza dell'n–esimo CPU burst
2. τ_{n+1} = valore stimato del prossimo CPU burst
3. α, $0 \le \alpha \le 1$
4. Si definisca :

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$$

esempio

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \qquad \alpha = 1/2$$



Effettivi	6	4	6	4	13	13	13	13
Stimati	10	8	6	6	5	9	11	12

Esempi di media esponenziale

- α =0
  - $\tau_{n+1} = \tau_n$
  - La storia recente non è presa in considerazione.
- α =1
  - $\tau_{n+1} = t_n$
  - Viene considerato soltanto l'ultimo CPU burst.
- Espandendo la formula si ottiene:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + \ldots + (1-\alpha)^j\,\alpha\, t_{n-j} + \ldots + (1-\alpha)^{n+1}\,\tau_0$$

- Poiché α e (1- α) sono entrambi minori o uguali ad 1, ciascun termine ha minor peso del suo predecessore.

Scheduling a priorità

- Un valore di priorità (intero) è associato a ciascun processo.
- La CPU viene allocata al processo con la priorità più alta
  - preemptive
  - non-preemptive
- ESEMPIO
  - **SJF** è uno scheduling a priorità dove la priorità è rappresentata dal successivo tempo di burst.

Priorita'

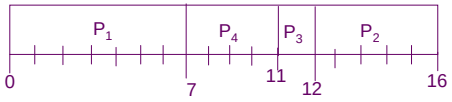
La priorita' e' un numero intero in un intervallo fissato
Es
- $0 \le p \le 20$
- $0 \le p \le 4096$

Ogni sistema ha il proprio modo di misurare la priorita'
➔ p=0 priorita' alta oppure
➔ p=0 priorita' bassa (es. Windows e Linux)

Esempio : priorita' 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

Problemi:

- SJF:
  - Forte disparita' tra processi corti e processi lunghi
- Priorita':
  - **Possibilita' di Starvation** (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.

Highest Response Ratio Next scheduling

- Scheduling senza prelazione a priorita' variabile
- E' un modo per realizzare l'aging
- La priorita' e' funzione di
  - Tempo di attesa
  - Tempo di burst

Favorisce i processi che hanno gia' atteso molto

- Esempio

$$Priorita = \frac{T_{attesa} + T_{burst}}{T_{burst}}$$

Favorisce i processi corti

p=1 minima priorita'

Esempio HRRN vs SJF

p=1 minima priorita'

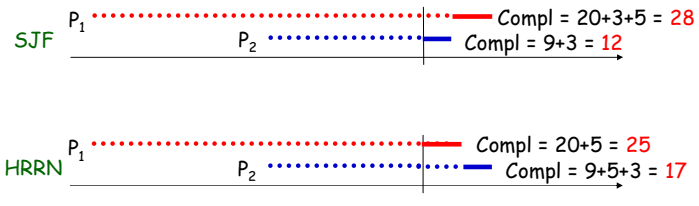
Processo	Tempo attesa	Tempo di burst
P_1	20	5
P_2	9	3

Processo	Priorita' HRRN
P_1	(20+5)/5=5
P_2	(9+3)/3 = 4

- SJF prima P_2 e poi P_1
- HRRN prima P_1 e poi P_2

Esempio HRRN vs SJF

Processo	Tempo attesa	Tempo di burst
P_1	20	5
P_2	9	3



Tempi medi di completamento

SJF = (28+12)/2 = 20 HRRN = (25+17)/2=21

problema

HRRN e' stato introdotto per superare l'eccessiva disparita' di trattamento tra processi corti e processi lunghi operata da SJF

Come misurare questa qualita' di HRRN?

La deviazione standard (o scarto quadratico medio) è una misura della dispersione di un insieme di numeri $x_1,\ldots, x_n$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} \quad con \quad \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

media

σ piccolo ➔ dati "vicini" tra loro
σ grande ➔ dati "lontani" tra loro

Esempio HRRN vs SJF

- SJF

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} = \sqrt{\frac{1}{2}(64+64)} = 8$$

- HRRN

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} = \sqrt{\frac{1}{2}(16+16)} = 4$$

La politica HRRN e' piu' equa
(i tempi di completamento sono piu' omogenei)

Scheduling Round Robin (RR)

- Scheduling con prelazione progettato appositamente per sistemi time sharing
- A ciascun processo viene allocata una piccola unità di tempo di CPU (*quanto di tempo* o *timeslice*), generalmente 10–100 millisecondi. Trascorso il quanto di tempo, il processo è forzato a rilasciare la CPU e accodato alla ready queue.
- Se ci 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.

Scheduling RR con quanto = 20

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

- Il diagramma di Gantt è:

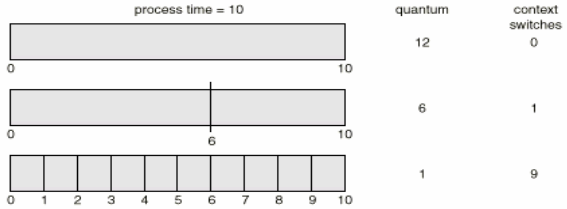
P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

- 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.

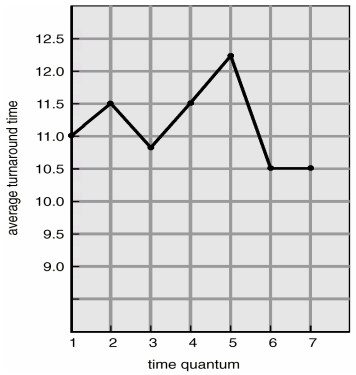
Quanto di tempo

- La grandezza del quanto di tempo determina il tempo di risposta delle richieste interattive
- Quanto di tempo grande
  - Processi in esecuzione per molto tempo
  - Di fatto diventa FCFS
- Quanto di tempo piccolo
  - Il sistema impiega piu' tempo nel cambio di contesto che nell'esecuzione dei programmi



Quanto di tempo e tempo di turnaround

Empiricamente: il quanto di tempo deve essere > dell'80% dei CPU burst.

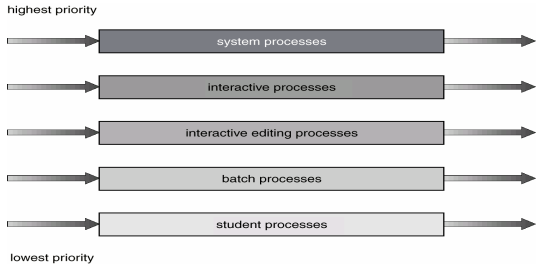


process	time
P_1	6
P_2	3
P_3	1
P_4	7

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

Scheduling con code multiple

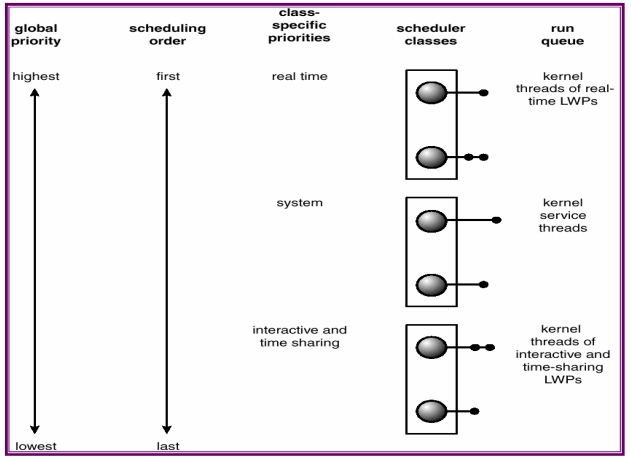
- La ready queue è suddivisa in code separate:
  - **foreground** (interattiva)
  - **background** (batch)
- Ciascuna coda ha il suo proprio algoritmo di scheduling:
  - foreground – **RR**
  - background – **FCFS**



Scheduling con code multiple

- È necessario uno scheduling tra code.
  - *Scheduling a priorità fissa*; es. serve tutti i processi in foreground poi in 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 in **RR**
    - 20% per background in **FCFS**

Solaris 2 Scheduling

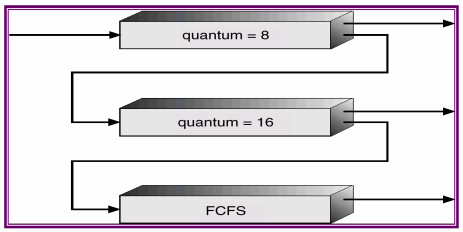


Code multiple con feedback

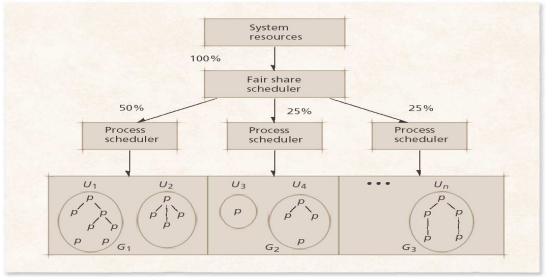
- Un processo può spostarsi fra le varie code; l'aging può essere implementato in questo modo.
- Lo scheduler con code multiple con feedback è definito dai seguenti parametri:
  - Numero di code
  - Algoritmi di scheduling per ciascuna coda
  - Metodo impiegato per determinare quando spostare un processo 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 richiede un servizio

Esempio 1

- Tre code: Q_0 – quanto di tempo di 8 millisecondi; Q_1 – quanto di tempo di 16 millisecondi; Q_2 – **FCFS**.
- Scheduling:
  - Un nuovo job viene immesso nella coda Q_0 che è servita **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 mosso nella coda Q_2.



Esempio 2: scheduler di UNIX



- UNIX usa uno scheduler chiamato Fair Share Scheduling
- a gruppi omogenei di processi (generici, di sistema) viene assegnata una priorita' iniziale (che vengono gestite con code multiple)
- Ogni coda e' gestita Round Robin
- scaduto il quanto di tempo si abbassa la priorita' e cambia coda

Cenni al real-time

I s.o. real time hanno algoritmi di scheduling differenti dai sistemi interattivi / time-sharing

Obiettivo primario: assicurare che un dato processo finisca entro i termini previsti (deadline)

In generale i s.o. real time gestiscono mansioni specifiche di durata fissata
- processi periodici (ad es. raccogliere informazioni sul traffico aereo ogni secondo)
- processi asincroni (ad es. in risposta ad un dato evento)

⇩

- facile prevedere la durata dell'esecuzione
- possibile usare algoritmi di scheduling a priorita' fissa

Earliest deadline first (EDF)

- algoritmo con prelazione che da' precedenza ai processi con deadline piu' vicino
- Obiettivo: soddisfare la deadline del maggior numero di processi
- Si puo' dimostrare che tale algoritmo minimizza l'eventuale ritardo del processo "piu' ritardatario"
- A volte non utilizzabile per la mancanza di supporto hardware (timer necessario per la prelazione)

Minimum laxity first (MLF)

Il rilassabilita' (laxity) e' una misura della non urgenza di un processo :

Ad es. se per un dato processo
T = tempo corrente = 5
D = deadline = 9
C = tempo di esecuzione rimanente = 3

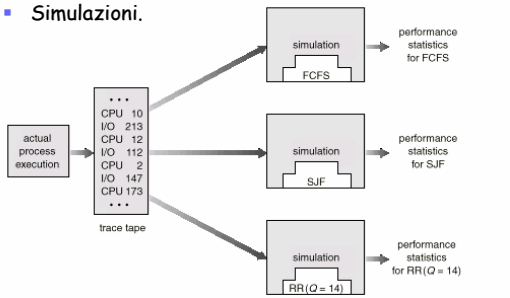
$L=(D-T) - C = 1$
e' il tempo che tale processo puo' attendere prima di essere mandato in esecuzione e non mancare la propria deadline

⇩

Nell'algoritmo MLF viene data
precedenza ai processi con la rilassabilita' minore

Valutazione degli algoritmi

- Modellazione deterministica — prende in considerazione un carico di lavoro predeterminato e definisce le prestazioni di ciascun algoritmo per tale carico di lavoro.
- Modelli di code — modelli statistici del comportamento del sistema.
- Simulazioni.



Valutazione di scheduler di CPU tramite simulazione