Sistemi Operativi A
Parte III - La gestione dell'unità centrale

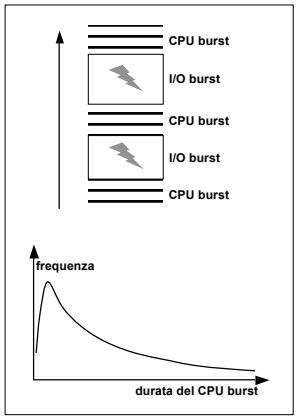
Augusto Celentano
Università Ca' Foscari Venezia
Corso di Laurea in Informatica

Scheduling dell'unità centrale

- In un sistema multiprogrammato l'esecuzione passa da un processo all'altro seguendo opportune strategie di gestione delle risorse
  - es. sovrapposizione della elaborazione di un processo con l'attesa da parte di altri
- La scelta di quale processo selezionare per l'esecuzione determina le politiche di scheduling
  - utilizzo del processore
  - throughput (numero di processi completati nell'unità di tempo)
  - tempo di risposta

Cicli di CPU - I/O

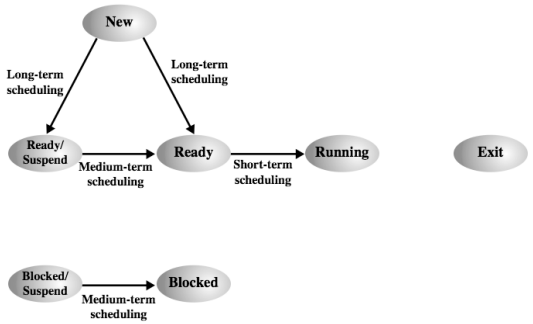
- L'esecuzione di un processo è costituita dall'alternanza ciclica tra due fasi:
  - esecuzione di istruzioni (*CPU burst*, breve)
  - attesa di eventi o operazioni esterne (*I/O burst*, lunga)
- La distribuzione dei *CPU burst* è nota statisticamente
  - i processi *I/O bound* hanno CPU burst molto brevi
  - i processi *CPU bound* hanno CPU burst più lunghi



Classificazione dei livelli di scheduling (1)

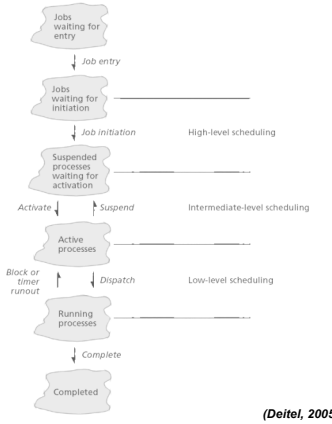
- A lungo termine (ad alto livello)
  - quale processo è ammesso a competere per le risorse
  - quale processo inizia l'esecuzione tra quelli che la richiedono
- A medio termine (a livello intermedio)
  - quale processo è ammesso a competere per l'uso del processore
  - quale processo resta in memoria o viene portato fuori memoria durante l'attesa o lo stato di pronto (*swap in* / *swap out*)
- A breve termine (a basso livello)
  - come vengono assegnate le priorità
  - quale processo pronto viene scelto per l'esecuzione

Classificazione dei livelli di scheduling (2)



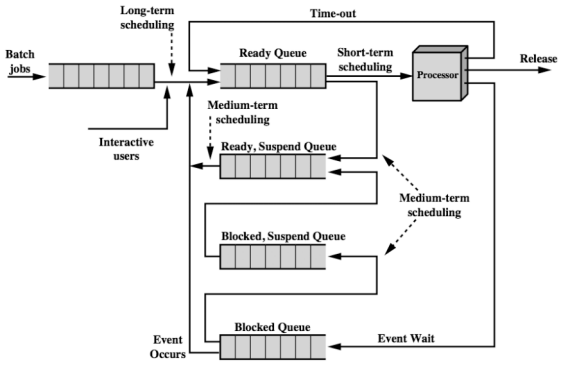
(Stallings, 2005)

Classificazione dei livelli di scheduling (2)



(Deitel, 2005)

Scheduling e gestione delle code



(Stallings, 2005)

Scheduling a lungo termine

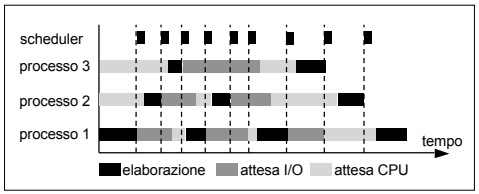
- Determina quali programmi vengono ammessi all'esecuzione
- Controlla il livello di multiprogrammazione
- L'ammissione di molti processi
  - riduce le probabilità che siano tutti in stato di attesa
  - utilizza complessivamente meglio l'unità centrale
  - … ma ogni processo ha a disposizione una minore percentuale di tempo di CPU
- Può bilanciare la presenza di processi CPU bound e I/O bound

Scheduling a medio termine

- Gestisce la permanenza in memoria dei processi correntemente non in esecuzione
  - soddisfa le esigenze delle multiprogrammazione
  - considera i tempi presumibili di attesa (attesa su un evento lento)
  - gestisce le priorità e le esigenze real-time
  - si appoggia al gestore della memoria centrale per l'esecuzione delle operazioni di swap

Scheduling a breve termine

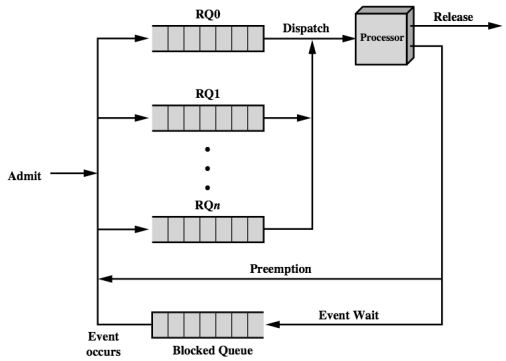
- Decide quale processo pronto va in esecuzione
  - scheduler di CPU
  - dispatcher
- Viene invocato su eventi che possono causare il passaggio ad un altro processo (politiche di scheduling)
  - interrupt di clock
  - interrupt di I/O
  - chiamate al sistema operativo
  - segnalazioni tra processi



Gestione delle priorità (1)

- La gestione delle priorità è implementata da un sistema di code multiple che ne rappresentano i diversi livelli
  - lo scheduler sceglie i processi da eseguire esaminando le code nell'ordine
  - i processi a priorità più bassa possono attendere un tempo indefinito (*starvation*)
  - le priorità possono essere modificate esaminando la storia passata di un processo per evitare fenomeni di *starvation*
- Gli algoritmi di scheduling si differenziano anche per come gestiscono le priorità

Gestione delle priorità (2)



(Stallings, 2005)

Pre-emption (1)

- I momenti in cui può intervenire lo scheduler a breve termine sono quattro:
  1. un processo passa da stato di esecuzione a stato di attesa: quale altro processo pronto andrà in esecuzione?
  2. un processo passa da stato di attesa a stato di pronto: con quali altri processi pronti compete?
  3. un processo passa da stato di esecuzione a stato di pronto (es. per una interruzione): tornerà in esecuzione subito o no?
  4. un processo termina: quale processo verrà eseguito?
- Nei casi 1 e 4 si *deve* scegliere un altro processo pronto; nei casi 2 e 3 si *può* scegliere un altro processo pronto, oppure si può lasciare o mandare in esecuzione il processo che ha cambiato stato

Pre-emption (2)

- La sospensione forzata dell'esecuzione di un processo per eseguirne un altro è detta *pre-emption*
- Le politiche di scheduling che si limitano ai casi 1 e 4 (esecuzione → attesa, esecuzione → terminazione) si dicono *non pre-emptive*
  - ad un processo in esecuzione non viene mai tolto di autorità il diritto a proseguire
- Quelle che intervengono anche nei casi 2 e 3 (attesa → pronto, esecuzione → pronto) si dicono *pre-emptive*
  - un processo in esecuzione può essere bloccato a favore di un altro processo

Politiche di gestione dell'unità centrale (1)

- Definiscono i principi in base ai quali il nucleo del sistema operativo assegna l'uso della unità centrale ai processi pronti
- Si possono dividere secondo tre criteri di classificazione
  - non pre-emptive vs. pre-emptive
  - senza priorità vs. con priorità
  - statiche vs. dinamiche
- In pratica utilizzano combinazioni dei tre criteri di classificazione
  - es. Unix (time sharing): pre-emptive, con priorità dinamiche

Politiche di gestione dell'unità centrale (2)

- Non pre-emptive
  - si basano sulla sospensione spontanea del processo in esecuzione
  - adatte per elaborazioni batch
- Pre-emptive
  - si basano sulla interruzione forzata del processo in esecuzione
  - soddisfano esigenze di priorità o di equità di ripartizione delle risorse
  - evitano il monopolio della CPU da parte di un processo CPU bound

Politiche di gestione dell'unità centrale (3)

- Senza priorità
  - considerano i processi equivalente sul piano dell'urgenza di esecuzione
  - si basano normalmente su strategie di ordinamento *First Come, First Served* (code *FIFO*, *First In First Out*)
- Con priorità
  - dividono i processi in classi secondo l'importanza o la criticità rispetto al tempo di esecuzione
  - necessarie nei sistemi con proprietà real-time e nei sistemi interattivi

Politiche di gestione dell'unità centrale (4)

- Statiche
  - un processo conserva nel tempo i suoi diritti di accesso all'unità centrale (priorità)
  - penalizza i processi a bassa priorità (*starvation*)
- Dinamiche
  - i processi modificano i propri diritti nel tempo basandosi sul comportamento passato o estrapolando quello futuro
  - bilancia le esigenze tra processi CPU bound e processi I/O bound

Indicatori di prestazioni (1)

- La misura delle prestazioni di un sistema di calcolo può essere effettuata tramite diversi indicatori e ha come scopo l'ottimizzazione della ripartizione delle risorse, in particolare l'utilizzo dell'unità di elaborazione
  - attività di CPU
  - livello di multiprogrammazione
  - tempo di attesa
  - tempo di turnaround
  - tempo di risposta
  - throughput
  - fairness

Indicatori di prestazioni (2)

- Attività di CPU
  - rapporto tra il tempo trascorso nella esecuzione dei processi di utente e il tempo trascorso nelle funzioni di S. O. e in attesa
- Livello di multiprogrammazione
  - numero di programmi contemporaneamente presenti in memoria centrale
- Tempo di attesa
  - tempo trascorso tra la richiesta di esecuzione e l'effettivo inizio dell'esecuzione (tempo trascorso da un processo nello stato di pronto)

Indicatori di prestazioni (3)

- Tempo di turnaround (batch)
  - tempo trascorso tra l'ingresso di un programma nel sistema e la fine dell'elaborazione; il tempo medio è la media dei tempi dei singoli programmi
- Tempo di risposta (interattivo)
  - tempo trascorso tra l'immissione di un comando e l'emissione della risposta
- Throughput (produttività)
  - lavoro svolto dal sistema nell'unità di tempo (numero di processi completati)
- Faimess
  - misura dell'omogeneità di trattamento tra processi diversi

Un confronto tra gli algoritmi di scheduling

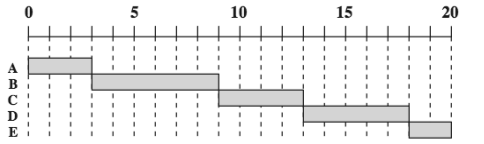
processo	tempo di arrivo	tempo di servizio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- tempo di arrivo = tempo di ingresso nello stato di pronto
- tempo di servizio = tempo di CPU richiesto in un CPU burst

First Come First Served (1)

- Esegue i processi nell'ordine in cui si trovano nella coda dei processi pronti
  - il prossimo processo servito è il processo più vecchio
  - non pre-emptive, con o senza priorità, statico
  - è l'algoritmo di base dei sistemi batch

processo	tempo di arrivo	tempo di servizio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



(Stallings, 2005)

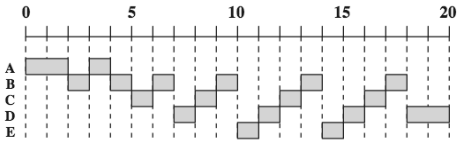
First Come First Served (2)

- Un processo che non esegue operazioni di I/O ha il monopolio della CPU
- E' un algoritmo che privilegia i processi *CPU bound*
  - un processo I/O bound deve attendere che un processo CPU bound finisca l'esecuzione di un lungo CPU burst
  - un processo I/O bound può attendere (pronto) anche quando l'operazione di I/O è terminata
  - il risultato è un'inefficiente utilizzo dei dispositivi di I/O (pause tra due richieste di operazioni di I/O)

Round Robin (1)

- Assegna ad ogni processo un quanto di tempo, scaduto il quale il processo è interrotto, rimesso in coda, e l'unità assegnata ad un altro processo
  - garantisce che tutti i processi avanzino in modo equilibrato
  - pre-emptive, con o senza priorità, statico o dinamico
  - è l'algoritmo di base dei sistemi time-sharing

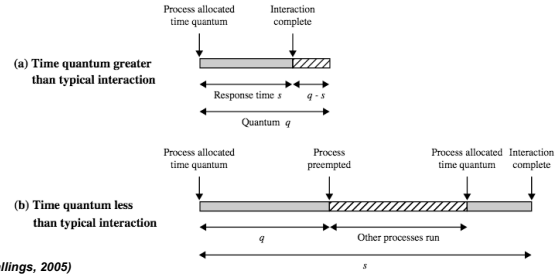
processo	tempo di arrivo	tempo di servizio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



(Stallings, 2005)

Round Robin (2)

- La scelta del quanto di tempo è critica
  - deve essere molto maggiore del tempo necessario a effettuare lo scheduling
  - deve essere maggiore del tempo medio necessario per eseguire un CPU burst nei processi I/O bound

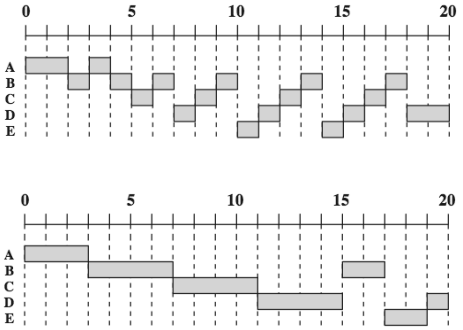


(Stallings, 2005)

Round Robin (3)

Round-Robin (RR), $q = 1$

processo	tempo di arrivo	tempo di servizio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



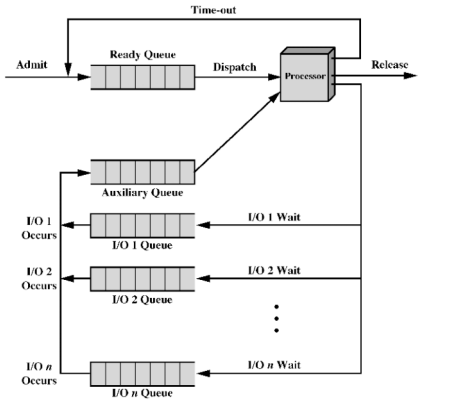
Round-Robin (RR), $q = 4$

(Stallings, 2005)

Round Robin (4)

- Privilegia i processi CPU bound
  - un processo I/O bound utilizza (spesso) l'unità centrale per un tempo minore del quanto assegnato, quindi va in stato di attesa
  - un processo CPU bound utilizza l'intero quanto assegnato, quindi va in stato di pronto (passa davanti ai processi in attesa)
- Le prestazioni dei processi I/O possono migliorare assegnando priorità più alte (RR virtuale)
  - un processo che completa un'operazione di I/O viene posto in una coda diversa, prioritaria rispetto alla coda dei processi pronti

Round Robin (5)

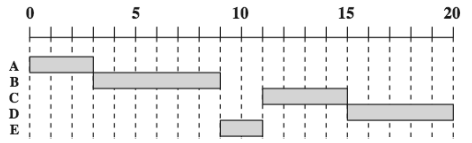


(Stallings, 2005)

Shortest Process Next (1)

- Esegue ad ogni turno il processo che prevedibilmente userà l'unità centrale per il tempo minore, prima di sospendersi o terminare
  - la valutazione del comportamento futuro è proiettata dal comportamento passato
  - non pre-emptive, con o senza priorità, dinamica
  - privilegia i processi I/O bound

processo	tempo di arrivo	tempo di servizio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



(Stallings, 2005)

Shortest Process Next (2)

- La predizione del comportamento futuro si basa sulla valutazione media del comportamento passato
  - Ti = tempo misurato dell'i-esimo CPU burst
  - Si = tempo predetto dell'i-esimo CPU burst

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i = \frac{1}{n}T_n + \frac{n-1}{n}S_n$$

- Questa soluzione calcola una media uniforme in cui la storia passata ha lo stesso peso di quella recente
  - non approssima bene la realtà
  - il comportamento recente è più influente di quello remoto

Shortest Process Next (3)

- La media esponenziale è più adeguata per calcolare una stima attendibile

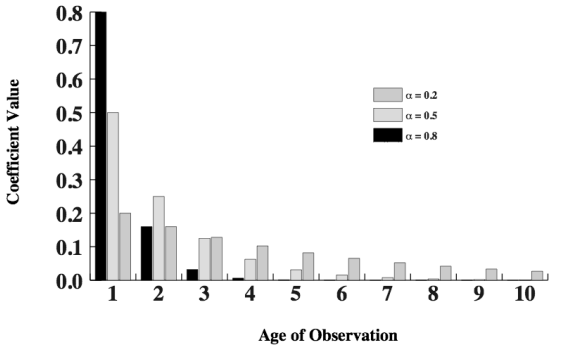
$$S_{n+1} = \alpha T_n + (1-\alpha)S_n$$

  - se $\alpha > 1/n$ i CPU burst più recenti hanno maggior peso
- L'espansione di questa formula mostra il decremento esponenziale dei termini successivi

$$S_{n+1} = \alpha T_n + (1-\alpha)\alpha T_{n-1} + \ldots (1-\alpha)^i \alpha T_{n-i} + \cdots + (1-\alpha)^n S_1$$

  - S_1 = 0 per privilegiare i nuovi processi

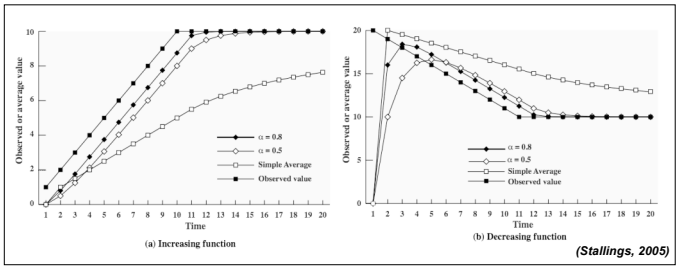
Shortest Process Next (4)



(Stallings, 2005)

Shortest Process Next (5)

- La media esponenziale si adatta meglio della media aritmetica ai cambiamenti di comportamento del processo



(Stallings, 2005)

Shortest Process Next (6)

- Implementa implicitamente una politica a priorità
  - i processi CPU bound hanno un servizio minore rispetto ai processi I/O bound
- I processi con CPU burst più lunghi sono soggetti a *starvation*
  - la presenza di molti processi interattivi mantiene basso il tempo di esecuzione predetto
- La mancanza di pre-emption non è adeguata per sistemi time-sharing
  - i processi CPU bound hanno priorità più bassa ma possono monopolizzare il sistema dopo un'assenza di processi interattivi

Real-time con priorità

- Divide i processi in classi di priorità
  - un processo in esecuzione usa l'unità centrale fino a che non si sospende da solo, a meno che non esista un processo pronto più prioritario, che assume la precedenza
  - pre-emptive su priorità, statico o dinamico
  - è alla base dei sistemi *hard real-time*
  - solitamente è combinato con altri algoritmi
  - le priorità sono divise in classi disgiunte per evitare il blocco dei processi critici del sistema operativo

A priorità variabile (1)

- Combina l'algoritmo Round Robin con una gestione dinamica della priorità dei processi (*multilevel feedback*)
  - i processi pronti sono posti in code (*ready queue*) con priorità differenti

$$P(rq_0) > P(rq_2) > \ldots > P(rq_n)$$

  - lo scheduler sceglie i processi pronti in ordine di priorità
  - i nuovi processi sono posti nella coda a priorità massima
  - scheduling pre-emptive con priorità dinamiche

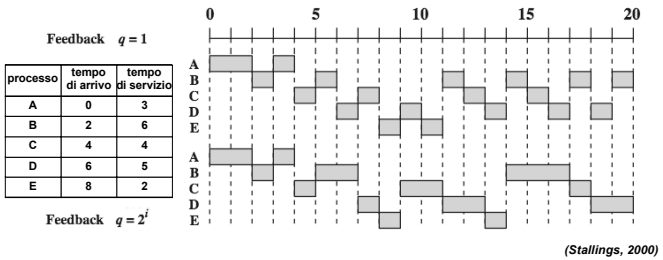
A priorità variabile (2)

- Le priorità sono modificate in funzione del comportamento dei processi
  - se un processo viene interrotto perché scade il suo quanto di tempo passa su una coda a priorità minore
  - se un processo si sospende prima dello scadere di un quanto dei tempo passa ad una coda di priorità superiore
  - favorisce i processi I/O bound
- I processi con CPU burst più lunghi sono soggetti a *starvation*
  - la modifica di priorità avviene anche se un processo resta a lungo nello stato pronto

A priorità variabile (3)

- Se il quanto di tempo è fisso i processi con CPU burst più lunghi avanzano lentamente
  - il quanto di tempo può essere modificato in funzione del livello di priorità

$$T(rq_i) = 2^{i-1}$$



(Stallings, 2000)

Fair Share Scheduling (1)

- In un sistema multiutente ogni utente può eseguire più processi
  - gli utenti hanno diritti diversi riguardo all'utilizzo delle risorse
  - utenti più attivi anche se meno importanti possono monopolizzare la macchina a scapito di utenti (temporaneamente) meno esigenti
  - le risorse libere possono essere distribuite tenendo conto delle risorse già allocate ai diversi processi di un utente
- Gli utenti possono a loro volta essere organizzati in gruppi
  - l'allocazione viene decisa sulle esigenze del gruppo e non del singolo processo

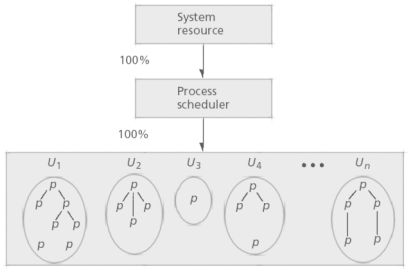
Fair Share Scheduling (2)

- ogni gruppo di processi (utente, gruppo di utenti) ha diritto ad un utilizzo *equo* della CPU (*fair share*)
  - la CPU è allocata ai processi considerando la loro appartenenza ai gruppi di utenti
  - i gruppi più numerosi ricevono meno risorse per ogni utente

$$P_j[i] = B_j + (1/2)\ CPU_j\ [i-1] + GCPU_k\ [i-1]/(4W_k)$$

Fair Share Scheduling (3)

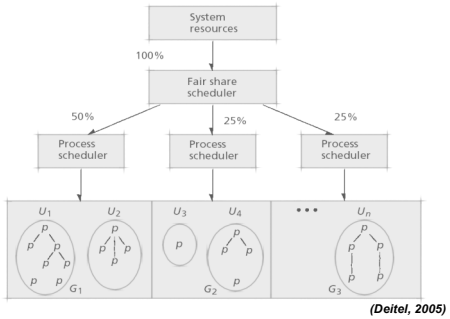
- In un sistema classico (Unix) le risorse sono allocate globalmente e suddivise tra i processi in base alle loro caratteristiche individuali



(Deitel, 2005)

Fair Share Scheduling (4)

- In un sistema fair share scheduling le risorse sono suddivise a priori tra gruppi di processi e successivamente allocate su base individuale



(Deitel, 2005)

Obiettivi degli algoritmi di scheduling (1)

- Per tutti i sistemi
  - *fairness*: assicurare ad ogni processo un equa ripartizione di risorse
  - *rispetto delle politiche* (*policy enforcement*): assicurare che l'algoritmo rispetti le politiche che si vogliono perseguire
  - *bilanciamento*: assicurare l'utilizzo di tutte le risorse del sistema
- Per i sistemi batch
  - *throughput*: massimizzare il numero di job eseguiti
  - *tempo di turnaround*: minimizzare il tempo tra la sottomissione di un lavoro e la sua esecuzione
  - *utilizzo della CPU*: mantenere la CPU occupata

Obiettivi degli algoritmi di scheduling (2)

- Per i sistemi interattivi
  - *tempo di risposta*: rispondere prontamente alla richieste di servizio (es. I/O)
  - *proporzionalità*: soddisfare le attese dell'utente
- Per i sistemi real-time:
  - *rispettare le scadenze*: evitare di perdere dati o eventi
  - *predicibilità*: prevedere il comportamento del sistema sotto le condizioni di carico previste per evitare il degrado delle prestazioni

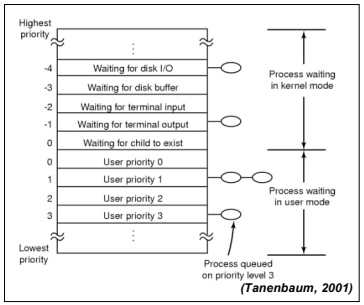
Qual è l'algoritmo migliore?

- La risposta dipende da
  - tipo di sistema (es. batch vs. interattivo vs. real-time)
  - peso relativo dei fattori di qualità (es. tempo di risposta vs. fairness vs. throughput)
  - carico del sistema (molto variabile)
  - metodo di valutazione utilizzato
  - …
- Quindi? …

Scheduling in Unix (1)

- A priorità variabile con algoritmo Round Robin per ogni livello di priorità
  - ogni processo ha una priorità di base + una parte variabile

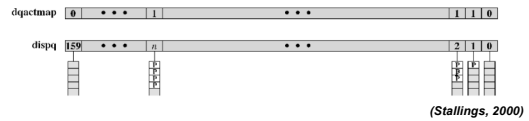
    $P_i = base_i + CPU_i + nice_i$
  - le priorità sono ricalcolate ogni secondo
  - i processi sono classificati in fasce di priorità senza sovrapposizioni
    - gestore memoria (swapper)
    - driver per I/O a blocchi
    - gestione file
    - driver per I/O a caratteri
    - processi di utente



(Tanenbaum, 2001)

Scheduling in Unix (2)

- Unix SVR4 introduce un meccanismo real-time dividendo i processi in tre classi di priorità
  - real-time (priorità = 159-100)
  - kernel (priorità = 99-60)
  - time sharing (priorità = 59-0)
- I processi kernel possono essere interrotti solo se le strutture dati del sistema sono coerenti o protette
- Le code dei processi pronti sono associate ad una struttura bitmap ad accesso veloce



(Stallings, 2000)

Scheduling in Linux (1)

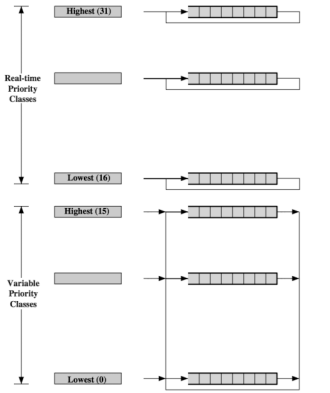
- Lo scheduling è basato sui thread, non sui processi
  - i thread sono implementati a livello kernel
- I thread sono divisi in tre classi
  - real-time FIFO (non soggetti a pre-emption)
  - real-time round robin (soggetti a pre-emption)
  - time sharing (a priorità minore rispetto ai primi due)
- Lo scheduling è basato sul concetto di *goodness* che misura *priorità* e *quanti* di tempo (= uso di CPU)
  - real-time → *goodness* = 1000 + priorità
  - timesharing & quanto > 0 → *goodness* = priorità + quanto
  - timesharing & quanto = 0 → *goodness* = 0

Scheduling in Linux (2)

- L'algoritmo di scheduling utilizza il valore di *goodness*
  - quando è necessaria una decisione si sceglie il thread con il più alto valore di *goodness*
  - durante l'esecuzione il valore di quanto è decrementato ad ogni evento di clock
- Lo scheduler interviene quando
  - il thread in esecuzione esaurisce il suo quanto di tempo (quanto = 0 → *goodness* = 0)
  - il thread si sospende su un'operazione di I/O o altro evento
  - un thread precedentemente sospeso con un valore maggiore di *goodness* diventa pronto (real-time)
- I thread I/O bound sono privilegiati
  - conservano parte dei quanti di tempo non utilizzati interamente

Scheduling in Windows 2000/XP (1)

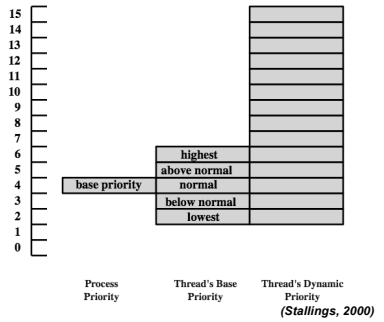
- Le priorità sono divise in due classi
  - priorità real-time
  - priorità variabile
- Lo scheduling è pre-emptive e guidato dalla priorità



(Stallings, 2005)

Scheduling in Windows 2000/XP (2)

- Le priorità dei thread è calcolata sulla priorità base del processo a cui appartengono
  - modificata da un fattore costante
  - ulteriormente variabile in funzione della dinamica del thread



(Stallings, 2000)