Sistemi Operativi Modulo I

Primo canale (A-L) e Teledidattica
A.A. 2019/2020
Corso di Laurea in Informatica

Lo Scheduling

Igor Melatti

Sapienza Università di Roma
Dipartimento di Informatica

SAPIENZA
UNIVERSITÀ DI ROMA

Roadmap

- Tipi di scheduler
- Algoritmi di scheduling
- Scheduling tradizionale di UNIX
- Scheduling su multiprocessore (cenni)

Scheduling

- Un sistema operativo deve allocare risorse tra diversi processi che ne fanno richiesta contemporaneamente
- Tra le diverse possibili risorse, c'è il tempo di esecuzione, che viene fornito da un processore
- Questa risorsa viene allocata tramite lo "scheduling" (possibile traduzione: pianificazione)

Scopo dello Scheduling

- Assegnare ad ogni processore i processi da eseguire, man mano che i processi stessi vengono creati e distrutti
- Tale obiettivo va raggiunto ottimizzando vari aspetti:
  - tempo di risposta
  - throughput
  - efficienza del processore

Obiettivi dello Scheduling

- Distribuire il tempo di esecuzione in modo *equo* (*fair*) tra i vari processi
- Evitare la starvation (letteralmente, morte per fame) dei processi
- Usare il processore in modo efficiente
- Avere un overhead basso
- Gestire le priorità dei processi quando necessario
  - ad esempio, quando ci sono vincoli di real time

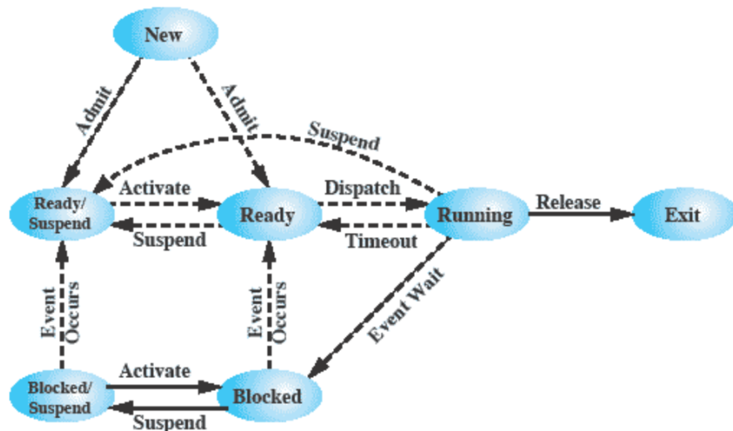
SAPIENZA
UNIVERSITÀ DI ROMA

Tipi di Scheduling

- Long-term scheduling (scheduling di lungo termine)
  - decide l'aggiunta ai processi da essere eseguiti
- Medium-term scheduling (scheduling di medio termine)
  - decide l'aggiunta ai processi che sono in memoria principale
- Short-term scheduling (scheduling di breve termine)
  - decide quale processo, tra quelli pronti, va eseguito da un processore
- I/O scheduling (scheduling per l'input/output)
  - decide a quale processo, tra quelli con una richiesta pendente per l'I/O, va assegnato il corrispondente dispositivo di I/O

Stati dei Processi

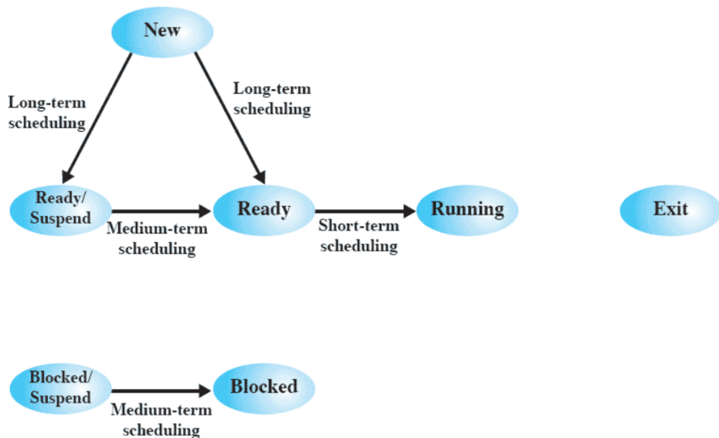
Modello a sette stati



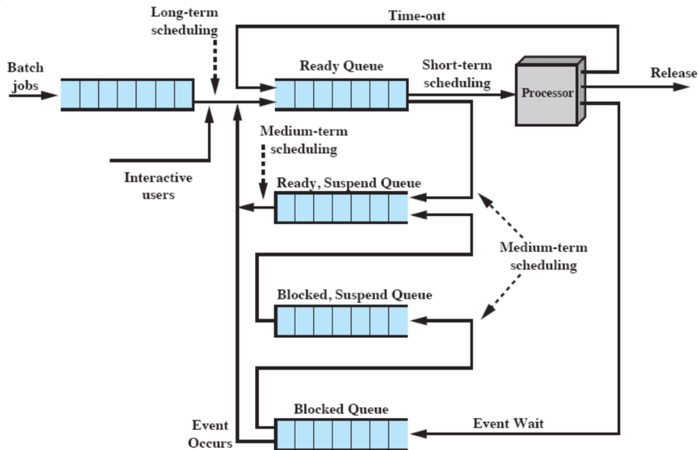
SAPIENZA
UNIVERSITÀ DI ROMA

Stati dei Processi e Scheduling

Queste sono le transizioni decise dai vari tipi di scheduler



Code dei Processi e Scheduling



SAPIENZA
UNIVERSITÀ DI ROMA

Long-Term Scheduling

- Decide quali programmi sono ammessi nel sistema per essere eseguiti
  - spesso è FIFO: il primo che arriva è il primo ad essere ammesso
  - altrettanto spesso, è un FIFO "corretto", tenendo conto di criteri come priorità, requisiti per l'I/O o tempo di esecuzione atteso
- Controlla il grado di multiprogrammazione
- Più processi ci sono, più è piccola la percentuale di tempo per cui ogni processo viene eseguito

Long-Term Scheduling

- Tipiche strategie:
  - i lavori batch vengono accodati e il LTS li prende man mano che lo ritiene “giusto”
  - i lavori interattivi vengono ammessi fino a “saturazione” del sistema
  - se si sa quali processi sono I/O bound e quali CPU-bound, può mantenere un giusto mix tra i 2 tipi
  - oppure, se si sa quali processi fanno richieste a quali dispositivi di I/O, fare in modo da bilanciare tali richieste
- Può essere chiamato in causa anche quando non ci sono nuovi processi
  - ad esempio, quando termina un processo
  - o quando alcuni processi sono idle da troppo tempo

Medium-Term Scheduling

- È parte della funzione di swapping per processi
  - ovvero del passaggio da memoria secondaria a principale e viceversa
- Il passaggio da memoria secondaria a principale è basato sulla necessità di gestire il grado di multiprogrammazione
- Ne riparleremo quando vedremo la gestione della memoria virtuale

Short-Term Scheduling

- Chiamato anche dispatcher
- È quello eseguito più frequentemente
- Invocato sulla base di eventi
  - interruzioni di clock
  - interruzioni di I/O
  - chiamate di sistema
  - segnali

Roadmap

- Tipi di scheduler
- Algoritmi di scheduling
- Scheduling tradizionale di UNIX
- Scheduling su multiprocessore (cenni)

Scopo dello Short-Term Scheduling

- Allocare tempo di esecuzione su un processore per ottimizzare il comportamento dell'intero sistema, dipendentemente da determinati indici prestazionali
- Per valutare una data politica di scheduling, occorre prima definire dei criteri

Criteri per lo Short-Term Scheduling: Utente contro Sistema

- Occorre distinguere tra criteri per l'utente e criteri per il sistema
- Per l'utente:
  - tempo di risposta
  - ovvero, il tempo che passa tra la richiesta di una computazione e la visualizzazione dell'output
- Per il sistema:
  - uso efficiente ed effettivo del processore

Criteri per lo Short-Term Scheduling: Prestazioni

- Occorre distinguere tra criteri correlati e non correlati alle prestazioni
- Quelli correlati alle prestazioni sono quantitativi e facili da misurare
  - tempo di risposta, throughput
- Quelli non correlati sono qualitativi e difficili da misurare
  - predicibilità, equità

Criteri Utente per lo Short-Term Scheduling

- Prestazionali:
  - **Turnaround time** (tempo di ritorno)
  - **Response time** (tempo di risposta)
  - **Deadline** (scadenza)
- Non prestazionali:
  - **Predictability** (predicibilità)

Criteri di Sistema per lo Short-Term Scheduling

- Prestazionali:
  - **Throughput** (volume di lavoro nel tempo)
  - **Processor utilization** (uso del processore)
- Non prestazionali:
  - **Fairness** (equità)
  - **Enforcing priorities** (gestione delle priorità)
  - **Balancing resources** (bilanciamento nell'uso delle risorse)

Turn-around Time

- Tempo tra la creazione (sottomissione) di un processo e il suo completamento
- Comprende i vari tempi di attesa (I/O, processore)
- Ok per un processo batch (non interattivo)

Response Time

- Tempo tra la sottomissione di una richiesta e l'inizio della risposta
- Ok per processi interattivi e nei quali si comincia a dare una risposta mentre ancora il processo non è finito
- Duplice obiettivo per lo scheduler
  - minimizzare il tempo di risposta medio
  - massimizzare il numero di utenti con un buon tempo di risposta

Deadline e Predicibilità

- Nei casi in cui si possono specificare scadenze, lo scheduler dovrebbe come prima cosa massimizzare il numero di scadenze rispettate
- Non deve esserci troppa variabilità nei tempi di risposta e/o di ritorno
- A meno che il sistema non sia completamente saturo...

SAPIENZA
UNIVERSITÀ DI ROMA

Throughput

- Massimizzare il numero di processi completati per unità di tempo
  - o che abbiano cominciato a produrre output (per gli interattivi)
- È una misura di quanto lavoro viene effettuato
- Dipende anche da quanto tempo richiede, in media, un processo

Utilizzo del Processore

- Percentuale di tempo in cui il processore viene utilizzato
  - il processore dev'essere idle il minor tempo possibile
  - servono processi ready...
- Utile per sistemi costosi, condivisi tra più utenti
- Meno utile per sistemi con un solo utente, o per sistemi real-time

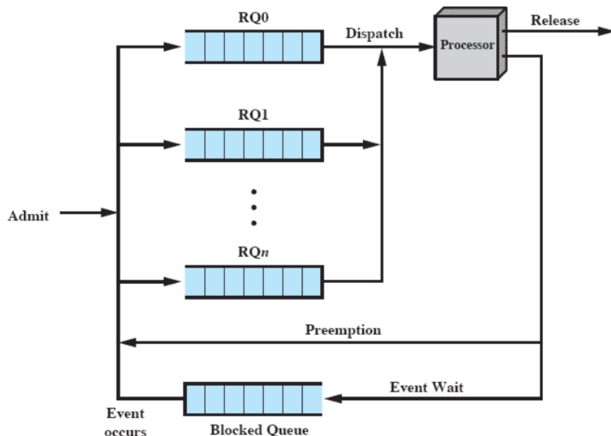
Bilanciamento delle risorse

- Lo scheduler deve far sì che le risorse del sistema siano usate il più possibile
- Processi che useranno meno le risorse attualmente più usate dovranno essere favoriti
- È un criterio che vale anche per gli scheduler di medio ed alto livello

Fairness e Priorità

- Se non ci sono indicazioni dagli utenti o dal sistema (ad es., priorità), tutti i processi devono essere trattati allo stesso modo
- Niente favoritismi: a tutti i processi dev'essere data la possibilità di andare in esecuzione
- Niente starvation
- Se invece ci sono priorità, lo scheduler deve favorire i processi a priorità più alta
- Occorrerà avere più code, una per ogni livello di priorità

Code per la Gestione delle Priorità



Priorità e Starvation

- Problema: un processo a bassa priorità potrebbe soffrire di starvation a causa di un altro processo a priorità più alta
- Soluzione: man mano che l'"età" del processo aumenta, la priorità cresce
  - o anche sulla base di quante volte ha potuto andare in esecuzione

Politiche di Scheduling

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s – e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

SAPIENZA
UNIVERSITÀ DI ROMA

Funzione di Selezione

- È quella che sceglie effettivamente il processo da mandare in esecuzione
- Se è basata sulle caratteristiche dell'esecuzione, i parametri da cui dipende sono:
  - w = tempo trascorso in attesa
  - e = tempo trascorso in esecuzione
  - s = tempo totale richiesto, incluso quello già servito (e)
    - inizialmente, $e = 0$, quindi s va o stimato o fornito come input insieme alla richiesta di creazione del processo

SAPIENZA
UNIVERSITÀ DI ROMA

Modalità di Decisione

- Specifica in quali istanti di tempo la funzione di selezione viene invocata
- Ci sono 2 possibilità:
  - preemptive
  - non-preemptive

Preemptive e Non-Preemptive

- Non-Preemptive:
  - se un processo è in esecuzione, allora arriva o fino a terminazione o fino ad una richiesta di I/O (o comunque ad una richiesta bloccante)
- Preemptive:
  - il sistema operativo può interrompere un processo in esecuzione anche se nessuna delle due precedenti condizioni è vera
  - in questo caso, il processo diverrà "pronto per l'esecuzione" (*ready*)
  - la preemption di un processo può avvenire o per l'arrivo di nuovi processi (appena forkati) o per un interrupt
    - interrupt di I/O: un processo blocked diventa ready
    - clock interrupt: periodico, evita che un processo monopolizzi il sistema

Roadmap

- Tipi di scheduler
- Algoritmi di scheduling
- Scheduling tradizionale di UNIX
- Scheduling su multiprocessore (cenni)

Politiche di Scheduling

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s – e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

SAPIENZA UNIVERSITÀ DI ROMA

Concetti Fondamentali

- Funzione di selezione
  - è quella che sceglie effettivamente il processo da mandare in esecuzione
- Modalità di decisione
  - non-preemptive: se un processo è in esecuzione, allora arriva o fino a terminazione o fino ad una richiesta di I/O
  - preemptive: il sistema operativo può interrompere un processo in esecuzione anche se nessuna delle due precedenti condizioni è vera

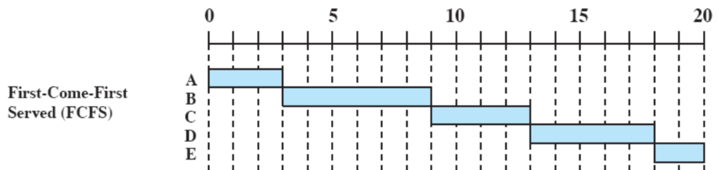
Scenario Comune di Esempio

5 processi batch

Processo	Tempo di arrivo	Tempo di esecuzione
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

FCFS: First Come First Served

- Tutti i processi sono aggiunti alla coda dei processi *ready*
- Quando un processo smette di essere eseguito, si passa al processo che ha aspettato di più nella coda ready finora
  - è non-preemptive, quindi solo se termina o richiede un I/O



FCFS: First Come First Served

- Un processo "corto" potrebbe dover attendere molto prima di essere eseguito
- Favorisce i processi che usano molto la CPU (*CPU-bound*)
  - una volta che un processo CPU-bound si è preso la CPU, non la rilascia più finché non ha finito

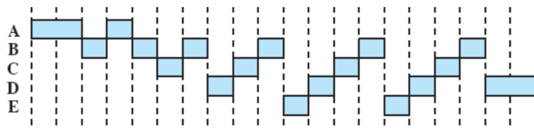
Scenario Comune di Esempio

5 processi batch

Processo	Tempo di arrivo	Tempo di esecuzione
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Round-Robin: un Po' per Ciascuno

- Usa la preemption, basandosi su un clock
- Talvolta chiamato *time slicing* (tempo a fette), perché ogni processo ha una fetta di tempo



Round-Robin: un Po' per Ciascuno

- Un'interruzione di clock viene generata ad intervalli periodici
- Quando l'interruzione di clock arriva, il processo attualmente in esecuzione viene rimesso nella coda dei ready
  - ovviamente, se il processo in esecuzione arriva ad un'istruzione di I/O prima dell'interruzione, allora viene spostato nella coda dei blocked
- Il prossimo processo ready nella coda viene selezionato

SAPIENZA
UNIVERSITÀ DI ROMA

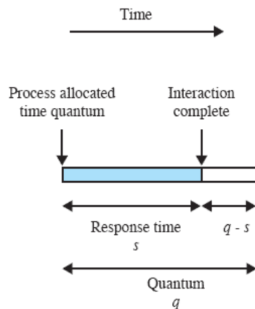
Round-Robin: un Po' per Ciascuno

T	Coda (prima)	Coda (dopo)	Esecuzione
0			A[3]
1			A[2]
2	B[6]	A[1]	B[6]
3	A[1]	B[5]	A[1]
4	B[5], C[4]	C[4]	B[5]
5	C[4]	B[4]	C[4]
6	B[4], D[5]	D[5], C[3]	B[4]
7	D[5], C[3]	C[3], B[3]	D[5]
8	C[3], B[3], E[2]	B[3], E[2], D[4]	C[3]
9	B[3], E[2], D[4]	E[2], D[4], C[2]	B[3]
10	E[2], D[4], C[2]	D[4], C[2], B[2]	E[2]

SAPIENZA UNIVERSITÀ DI ROMA

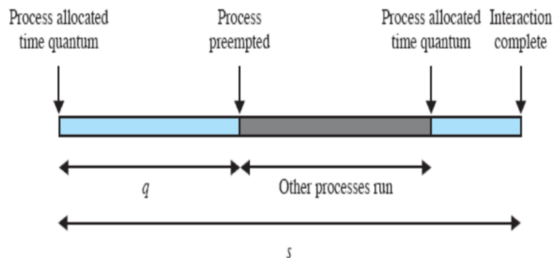
Misura del Quanto di Tempo per la Preemption

Se maggiore del tipico tempo di interazione: OK



Misura del Quanto di Tempo per la Preemption

Se minore del tipico tempo di interazione: non ottimale



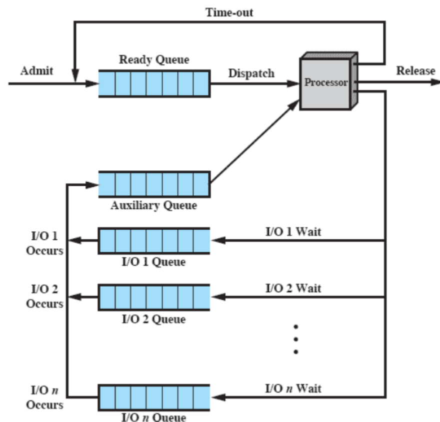
Misura del Quanto di Tempo per la Preemption

- Dev'essere di poco più lugo del "tipico" tempo di interazione per un processo
  - da calcolare statisticamente...
  - in realtà non un grosso problema nei sistemi operativi moderni: per i job interattivi, ci sono gli interrupt (vedere fine della lezione)
- Ma se lo si fa troppo lungo, potrebbe durare più del tipico processo
- ... e il round robin degenera in FCFS
- Insomma: più lungo possibile, ma senza farlo degenerare in FCFS

CPU-bound vs I/O bound

- Con il round-robin, i processi CPU-bound sono favoriti
  - usano tutto (o quasi) il loro quanto di tempo
- Invece, gli I/O bound ne usano solo una porzione
  - fino alla richiesta di I/O
- Non equo, oltre che inefficiente per l'I/O
  - aumenta anche la variabilità della risposta: non predicibile
- Soluzione: round-robin virtuale
  - dopo un completamento di I/O, il processo non va in coda ai ready, ma va in una nuova coda che ha priorità su quella dei ready
  - però, solo per la porzione del quanto che ancora gli rimaneva da completare (al più...)
  - migliora la fairness del round-robin semplice

Round-Robin Virtuale



SAPIENZA
UNIVERSITÀ DI ROMA

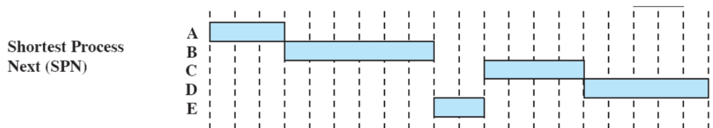
Scenario Comune di Esempio

5 processi batch

Processo	Tempo di arrivo	Tempo di esecuzione
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

SPN: Shortest Process Next

- Letteralmente: il prossimo processo da mandare in esecuzione è quello più breve
- Per "breve" si intende quello il cui tempo di esecuzione stimato è minore
  - tra quelli ready, ovviamente
- Quindi i processi corti scavalcano quelli lunghi
- Senza preemption



SPN: Shortest Process Next

- La predicibilità dei processi lunghi è ridotta
  - ovvero, è più difficile dire quando andranno in esecuzione: dipende molto da come si comportano gli altri processi nel sistema
- Se il tempo di esecuzione stimato si rivela inesatto, il sistema operativo può abortire il processo
- I processi lunghi potrebbero soffrire di starvation

SAPIENZA
UNIVERSITÀ DI ROMA

SPN: Come Stimare il Tempo di Esecuzione?

- In alcuni sistemi ci sono processi (sia batch che interattivi) che sono eseguiti svariate volte
- Si usa il passato (T_i) per predire il futuro (S_i)

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

$$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n$$

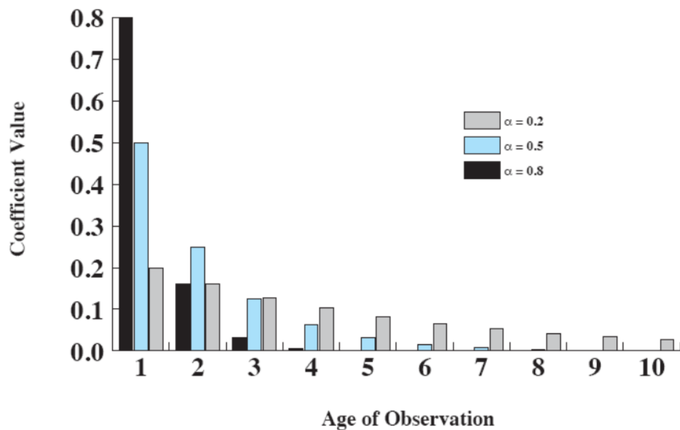
SPN: Come Stimare il Tempo di Esecuzione?

- La formula precedente dà lo stesso peso a tutte le istanze
- Meglio far pesare di più quelle più recenti: Exponential averaging

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n, 0 < \alpha < 1$$

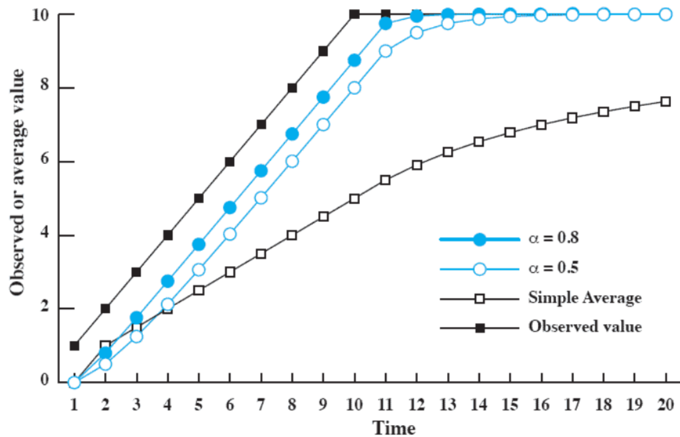
$$S_{n+1} = \alpha T_n + \ldots + \alpha (1 - \alpha)^i T_{n-i} + \ldots + (1 - \alpha)^n S_1$$

Exponential Averaging: Coefficienti

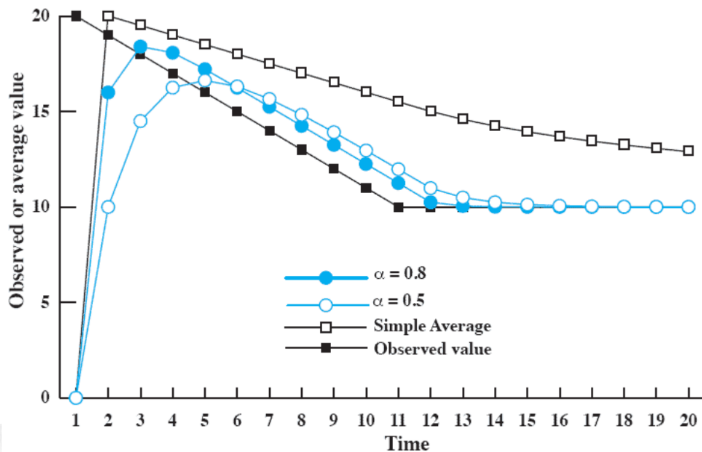


SAPIENZA
UNIVERSITÀ DI ROMA

Exponential Averaging: Esempio 1



Exponential Averaging: Esempio 2



SAPIENZA
UNIVERSITÀ DI ROMA

Scenario Comune di Esempio

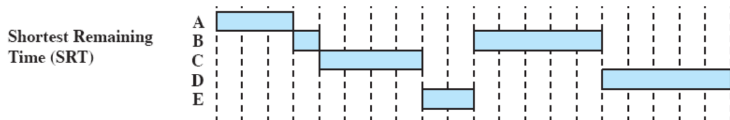
5 processi batch

Processo	Tempo di arrivo	Tempo di esecuzione
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

SAPIENZA
UNIVERSITÀ DI ROMA

SRT: Shortest Remaining Time

- Come SPN, ma preemptive
  - non con time quantum: un processo può essere interrotto solo quando ne arriva uno nuovo, appena creato
  - (o se fa un I/O bloccante)
- Stima il tempo rimanente richiesto per l'esecuzione, e prende quello più breve



Scenario Comune di Esempio

5 processi batch

Processo	**Tempo di arrivo**	**Tempo di esecuzione**
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

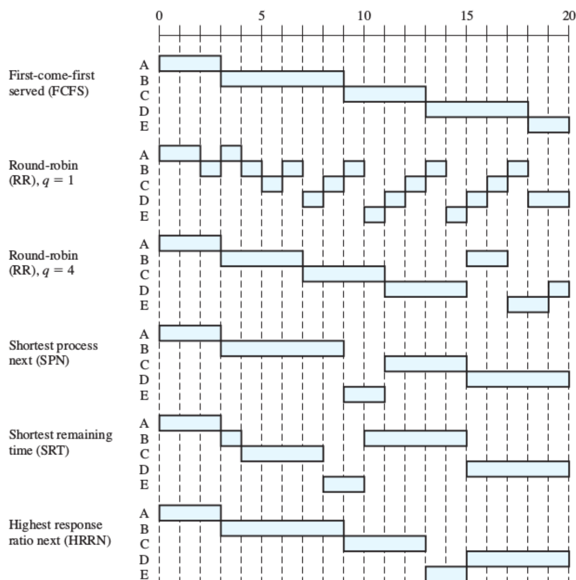
HRRN: Highest Response Ratio Next

- Massimizza il seguente rapporto

$$\frac{w+s}{s} = \frac{\text{tempo trascorso in attesa} + \text{tempo totale richiesto}}{\text{tempo totale richiesto}}$$

SAPIENZA
UNIVERSITÀ DI ROMA

Confronto



SAPIENZA UNIVERSITÀ DI ROMA

Roadmap

- Tipi di scheduler
- Algoritmi di scheduling
- Scheduling tradizionale di UNIX
- Scheduling su multiprocessore (cenni)

SAPIENZA
UNIVERSITÀ DI ROMA

Scheduling tradizionale di UNIX

- Combina priorità e round robin
- Un processo resta in esecuzione per al massimo un secondo, a meno che non termini o si blocchi
- Diverse code, a seconda della priorità; all'interno di ciascuna coda, si fa round robin
- Le priorità vengono ricalcolate ogni secondo: più un processo resta in esecuzione, più viene spinto in coda a minore priorità (feedback)
- Le priorità iniziali sono basate sul tipo di processo
  - swapper (alta)
  - controllo di un dispositivo di I/O a blocchi
  - gestione di file
  - controllo di un dispositivo di I/O a caratteri
  - processi utente (basso)

Formula di Scheduling

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

- $CPU_j(i)$ è una misura di quanto il processo j ha usato il processore nell'intervallo i, con exponential averaging dei tempi passati
- Per i running, $CPU_j(i)$ viene incrementato di 1 ogni $\frac{1}{60}$ di secondo
- $P_j(i)$ è la priorità del processo j all'inizio di i (più basso è il valore, più alta la priorità)
- $Base_j$: vedere slide precedente
- $nice_j$: lo stesso processo si può "declassare" come avente bassa priorità

Sapienza Università di Roma

Esempio di Scheduling su UNIX

$$Base_A = Base_B = Base_C = 60, nice_A = nice_B = nice_C = 0$$

Time	Process A Priority	Process A CPU count	Process B Priority	Process B CPU count	Process C Priority	Process C CPU count
0	60	0 1 2 • • 60	60	0	60	0
1	75	30	60	0 1 2 • • 60	60	0
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

Colored rectangle represents executing process

Roadmap

- Tipi di scheduler
- Algoritmi di scheduling
- Scheduling tradizionale di UNIX
- Scheduling su multiprocessore (cenni)

Architetture Multiprocessore

- Cluster
  - ogni processore ha la sua RAM, connessione con rete locale superveloce
- Processori specializzati
  - ad esempio, ogni I/O device ha un suo processore
- Multi-processore e/o multi-core
  - condividono la RAM
  - un solo SO controlla tutto
  - ci concentriamo solo su questi

Architetture Multiprocessore

- Cluster
  - ogni processore ha la sua RAM, connessione con rete locale superveloce
- Processori specializzati
  - ad esempio, ogni I/O device ha un suo processore
- Multi-processore e/o multi-core
  - condividono la RAM
  - un solo SO controlla tutto
  - ci concentriamo solo su questi

Scheduler su Architetture Multiprocessore

- Assegnamento statico
  - quando un processo viene creato, gli viene assegnato un processore
  - per tutta la sua durata, andrà in esecuzione su quel processore
  - si può usare uno scheduler (come quelli visti in precedenza) per ogni processore
  - vantaggi: semplice da realizzare, poco overhead
  - svantaggi: un processore può rimanere idle

Scheduler su Architetture Multiprocessore

- Assegnamento dinamico
  - per migliorare lo svantaggio dello statico, un processo, nel corso della sua vita, potrà essere eseguito su diversi processori
  - ragionevole, ma complicato da realizzare
- Il SO potrebbe essere sempre eseguito su un processore fisso
  - più semplice da realizzare
  - solo i processi utente possono "vagare"
  - svantaggio: può diventare il bottleneck
  - altro svantaggio: se il sistema potrebbe funzionare con una failure di un processore, se fallisce quello del SO cade tutto
- Il SO viene eseguito sul processore che capita
  - ottima flessibilità, ma richiede più overhead per gestire il SO "mobile"

Scheduling in Linux

- Cerca la velocità di esecuzione, tramite semplicità di implementazione
- Niente *long-term* né *medium-term* scheduler!
- Un embrione del long-term c'è
  - quando un processo ne crea un altro, questo viene o aggiunto alla runqueue appropriata, o non viene creato affatto
  - ma se non viene creato, è perché non c'è abbastanza memoria (neanche virtuale!) per farlo
- Per il medium-term, ci torneremo quando parleremo di gestione della memoria

Scheduling in Linux

- Ci sono le *runqueues* e le *wait queues*
- Le wait queues sono proprio le code in cui i processi sono messi in attesa quando fanno una richiesta che implichi attesa
  - ad esempio, una richiesta di I/O
- Le runqueues sono quelle da cui pesca il dispatcher (short-term scheduler)

Scheduling in Linux

- Essenzialmente derivato da quello di UNIX: preemptive a priorità *dinamica*
  - decrescente man mano che un processo viene eseguito
  - crescente man mano che un processo non viene eseguito
- Ma con importanti correzioni per:
  1. essere veloce, ed operare quasi in $O(1)$
  2. servire nel modo appropriato i processi real-time, se ci sono
- Linux istruisce l'hardware di mandare un timer interrupt ogni 1 ms
  - più lungo: problemi per applicazioni real-time
    - ma per architetture più vecchie deve salire a 10 ms
  - più corto: arrivano troppi interrupt, troppo tempo speso in Kernel Mode
  - e quindi di meno in user mode: sistema operativo non più conveniente...
- Il quanto di tempo per ciascun processo è quindi multiplo di 1 ms

Scheduling in Linux

- Tre tipi di processi
  1. Interattivi
     - non appena si agisce sul mouse o sulla tastiera, è importante dare loro la CPU in 150 ms al massimo
     - altrimenti l'utente percepisce che Linux non sta facendo il suo dovere
     - shell, GUI, ...
  2. Batch (non interattivi)
     - tipicamente penalizzati dallo scheduler: l'utente è disposto ad aspettare un po' di più
     - compilazioni, computazioni scientifiche, ...
  3. Real-time
     - gli unici riconosciuti come tali da Linux: il loro codice sorgente usa la system call `sched_setscheduler`
     - per gli altri, Linux usa un'euristica (ma le versioni più recenti hanno pure `sched_stscheduler` per loro)
     - riproduttori di audio/video, controllori, ... ma normalmente usati solo dai KLT di sistema
- Tutti possono essere sia CPU-bound che I/O-bound

SAPIENZA Università di Roma

Scheduling in Linux

- Ci sono 3 classi di scheduling
  - SCHED_FIFO e SCHED_RR fanno riferimento ai processi real-time
  - SCHED_OTHER tutti gli altri processi
  - versioni più recenti del kernel hanno anche altre classi, ma noi ci fermiamo qui
- Prima si eseguono i processi in SCHED_FIFO o SCHED_RR, poi quelli SCHED_OTHER
  - le prime 2 classi hanno un livello di priorità da 1 a 99, la terza da 100 a 139
  - quindi ci sono 140 runqueues
    - c'è anche una priorità 0 usata per casi particolari
    - in realtà, 140 per ogni CPU
  - si passa dal livello n al livello al livello $n + 1$ solo se o non ci sono processi in n, o nessun processo in n è in RUNNING

Scheduling in Linux

- La preemption può essere dovuta a 2 casi:
  - si esaurisce il quanto di tempo del processo attualmente in esecuzione
  - un altro processo passa da uno degli stati blocked a `RUNNING`
- Molto spesso, il processo che è appena diventato eseguibile verrà effettivamente eseguito dal processore
  - a seconda di quante CPU ci sono, può soppiantare il processo precedente
  - questo perché probabilmente si tratta di un processo interattivo, cui bisogna dare precedenza
  - esempio: editor di testo vs. compilatore
  - ogni mossa del mouse o tasto premuto è un interrupt, che causa una chiamata allo schedulatore
  - e probabilmente verrà data la precedenza all'editor
  - (tanto, la risposta ad un tasto premuto è spesso solo l'echo del carattere...)

Scheduling in Linux: Regole Generali

- Un processo SCHED_FIFO viene non solo preempted, ma anche rimesso in coda solo se:
  - si blocca per I/O (o rilascia volontariamente la CPU)
  - un altro processo passa da uno degli stati blocked a RUNNING, ed ha priorità più alta
- Altrimenti, non lo ferma nessuno
- Tutti gli altri processi vanno a quanti di tempo, compreso SCHED_RR
  - quindi, oltre a quanto detto sopra, un processo SCHED_RR viene rimesso in coda se esaurisce il suo quanto di tempo
  - RR sta per round-robin...
- I processi real-time non cambiano mai la priorità
- I processi SCHED_OTHER, sì
  - meccanismo simile allo UNIX tradizionale: priorità decrescente
  - con round-robin virtuale per processi ad uguale priorità
- Infine, per sistemi con CPU multiple c'è una routine periodica che ridistribuisce il carico, se necessario

SAPIENZA
UNIVERSITÀ DI ROMA

Scheduling in Linux: Regole Generali

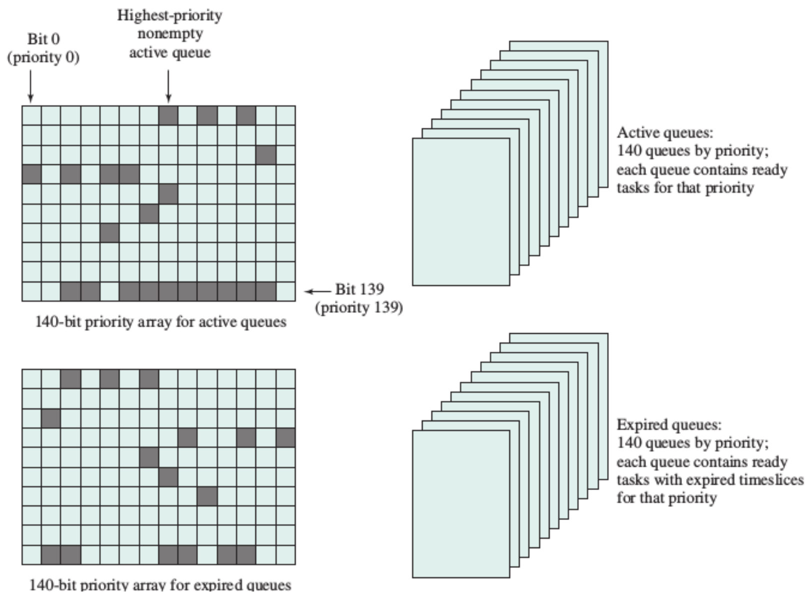
A	Minimum
B	Middle
C	Middle
D	Maximum

(a) Relative thread priorities

(b) Flow with FIFO scheduling

(c) Flow with RR scheduling

Scheduling in Linux: Regole Generali



SAPIENZA
UNIVERSITÀ DI ROMA