6 - Scheduling CPU

sabato 26 marzo 2022 11:54

SCHEDULING

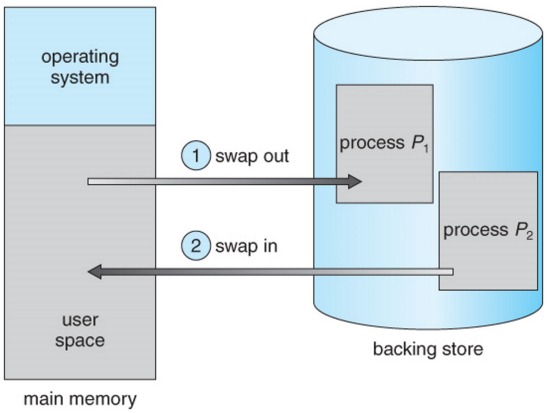
•Scheduling: assegnazione di attività nel tempo
•Lo scheduling ci è necessario per regolare:
-l'ammissione dei processi nella memoria
-l'ammissione dei processi nella CPU

TIPI DI SCHEDULER

•**Scheduler a lungo termine (job scheduler):** seleziona quali processi spostare in memoria
•**Scheduler a breve termine (CPU scheduler):** seleziona quali processi devono essere eseguiti dalla CPU
•**Scheduler a medio termine:** si occupa di swapping nella memoria virtuale

MEMORIA VIRTUALE

•Memoria virtuale: parte del disco che viene usata temporeaneamente come RAM
-questa parte del disco viene anche chiamata backing store
•**La CPU comunica SOLO con la RAM fisica**
-se un processo è nel backing store, devo passarlo nella RAM tramite swap-in (swap-out fa il contrario)
•A "swappare" i processi se ne occupa lo scheduler a medio termine



DISPATCHER

•Lo scheduler si occupa solo di scegliere i processi da mettere nella CPU, **a effettivamente spostarli se ne occupa il dispatcher**
•Il dispatcher deve essere quanto più rapido possibile

BURST

•Burst: sequenza
•Ogni processo può essere visto come una serie di CPU burst e I/O burst
-Solitamente ci sono tanti burst ma tutti molto brevi (pochi sono lunghi)
•Vengono utilizzati per decidere le politiche di scheduling

PRELAZIONE (PREEMPTION)

•Prelazione: rilascio forzato della CPU
•**Scheduling senza prelazione (non-preemptive):** il processo non lascia la CPU finchè non finisce il burst
•**Scheduling con prelazione (preemptive):** il processo può essere forzato a lasciare la CPU prima di finire il burst

ESEMPI REALI

•**Non-preemptive:** sono dal salumiere e anche se devo prendere solo 1 etto di prosciutto, devo aspettare la signora davanti a me che prende 5 etti di ogni salume
•**Preemptive:** sono al pronto soccorso, se arriva una persona gravemente ferita, devo lasciarla passare

METRICHE DI SCHEDULING

Per valutare una algoritmo di scheduling si usano le seguenti metriche:

- **Utilizzo della CPU:** percentuale di utilizzo media della CPU
- **Throughput:** numero di processi completati nell'unità di tempo

- Waiting time t_w: tempo speso nella ready queue
- Response time t_r: tempo compreso tra l'arrivo nella ready queue e la prima esecuzione (dispatch)
- Turnaround time t_t: tempo totale dall'inizio alla fine del processo = tempo di esecuzione (CPU burst) + tempo di attesa

NOTA: negli algoritmi non-preemptive, tempo di attesa e tempo di risposta sono uguali

ALGORITMI DI SCHEDULING

FIRST-COME, FIRST-SERVER (FCFS)

- **Non-preemptive**
- Funziona come un FIFO (primo processo ad entrare è il primo ad essere servito)
- Problemi:
  -performance variabili in base all'arrivo dei processi
  -processi CPU burst brevi vengono ritardati da processi CPU lunghe

Processo	Tempo di arrivo	CPU burst
P1	0	24
P2	2	3
P3	4	3

Processo	T_r	T_w	T_t
P1	0	0	24
P2	22	22	25
P3	23	23	26

Tempo	0	24	27	30
P1				
P2				
P3				

- Tempo di attesa medio
  – Tw_medio = (0+22+23)/3 = 15

Processo	Tempo di arrivo	CPU burst
P1	4	24
P2	0	3
P3	2	3

Processo	T_r	T_w	T_t
P1	2	2	26
P2	0	0	3
P3	1	1	4

Tempo	0	3	6	30
P1				
P2				
P3				

- Tempo di attesa medio
  – Tw_medio = (2+0+1)/3 = 1 (molto meglio!)

SHORTEST-JOB-FIRST (SJF)

- **Sia preemptive che non-preemptive**
- Viene selezionato il processo con CPU burst più breve
  -nel caso preemptive, se arriva un processo con CPU burst più breve del tempo rimanente a quello in esecuzione, quest'ultimo viene rimosso per fare spazio a quello appena arrivato
- Problemi:
  -Starvation: un processo con CPU burst molto lungo potrebbe non arrivare mai alla CPU

Non-preemptive

Processo	Tempo di arrivo	CPU burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Processo	T_r	T_w	T_t
P1	0	0	7
P2	6	6	10
P3	3	3	4
P4	7	7	11

Tempo	0	7	8	12	16
P1					
P2					
P3					
P4					

Preemptive

Processo	Tempo di arrivo	CPU burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Processo	T_r	T_w	T_t
P1	0	9	16
P2	0	1	5
P3	0	0	1
P4	2	2	6

Tempo	0	2	4	5	7	11	16
P1							
P2							
P3							
P4							

SCHEDULING A PRIORITÀ

- **Sia preemptive che non-preemptive**
- Viene data una priorità ad ogni processo
  -la CPU viene assegnata al processo con priorità più alta

-la priorità viene assegnata in base a diversi fattori interni (es. tempo) ed esterni (es. motivi politici)

•Problemi:

-Starvation: un processo con bassa priorità potrebbe non arrivare mai alla CPU (si può risolvere aumentando la priorità col tempo)

Proc.	T. di arrivo	Pr.	CPU burst
P1	1	3	10
P2	0	1	1
P3	2	3	2
P4	0	4	1
P5	1	2	5

Processo	T_r	T_w	T_t
P1	5	5	15
P2	0	0	1
P3	14	14	16
P4	18	18	19
P5	0	0	5

Tempo	0	1	6	16	18	19
P1						
P2						
P3						
P4						
P5						

HIGHER RESPONSE RATION NEXT (HRRN)

•**Non-preemptive**

•Viene data una priorità ad ogni processo attraverso questa formula:

•Vengono favoriti processi che

-si completano in poco tempo

-hanno aspettato molto

Proc.	T. di arrivo	CPU burst
P1	1	10
P2	0	2
P3	2	2
P4	2	1
P5	1	5

Calcolo priorità R (termine processo)				
Proc.	t=0	t=2	t=7	t=8
P1	-	1+1/10	1+6/10	1+7/10
P2	1	-	-	-
P3	-	1+0/2	1+5/2	1+6/2
P4	-	1+0/1	1+5/1	-
P5	-	1+1/5	-	-

Processo	T_r	T_w	T_t
P1	9	9	19
P2	0	0	2
P3	6	6	8
P4	5	5	6
P5	1	1	6

Tempo	0	2	7	8	10	19
P1						
P2						
P3						
P4						
P5						

ROUND ROBIN (RR)

•**Preemptive**

•Ad ogni processo viene dato un quanto (10-100 millisecondi) di tempo della CPU

•In base al quanto:

-q grande -> funziona come FCFS

-q piccolo -> troppo overhead dovuto ai vari context switch

•Se un processo non termina durante il suo quanto, verrà messo in fondo alla ready queue in attesa del suo turno

Proc. (*q*=2)	T. di arrivo	CPU burst
P1	0	5
P2	0	1
P3	0	7
P4	0	2

Processo	T_r	T_w	T_t
P1	0	7	12
P2	2	2	3
P3	3	8	16
P4	5	5	7

Tempo (*q*=2)	0	2	3	5	7	9	11	12	14	15
In esecuzione	P1	P2	P3	P4	P1	P3	P1	P3	P3	
Nella ready queue	P2	P3	P4	P1	P3	P1	P3			
	P3	P4	P1	P3						
	P4	P1								

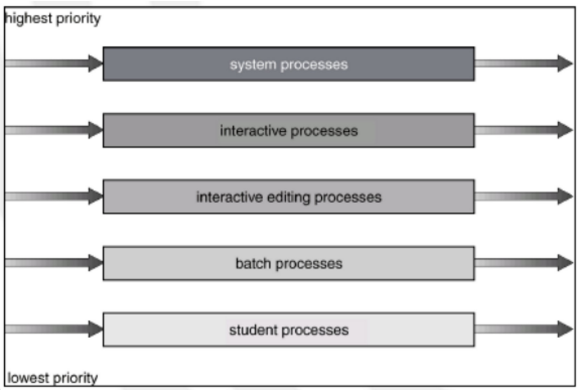
CODE MULTI-LIVELLO

•Per ottimizzare i tempi, non viene usata un'unica coda con un unico algoritmo, ma **diverse code dotate ognuna di priorità e algoritmi di scheduling diversi**
-es. 1: una coda per i processi in foreground (interattivi)
-es. 2: una coda per i processi in background (batch)

Diventa necessario un algoritmo per lo scheduling fra le code:
•**Scheduling a priorità fissa:**
-ogni coda ha una sua priorità fissa (i processi entrano in una coda in base alla loro priorità)
-servo prima tutti i processi di sistema, poi quelli in foreground e poi quelli in background
-Starvation: i processi in background potrebbero non essere mai serviti
•**Scheduling basato su time slice:**
-ogni coda ottiene un quanto del tempo di CPU
-le code con priorità più alta ottengono una percentuale di utilizzo maggiore

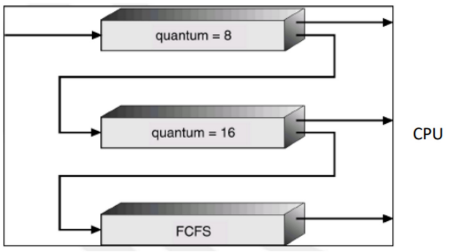
CODE MULTI-LIVELLO CON FEEDBACK

•**Code multilivello classiche:**
-i processi non possono cambiare coda
-problemi di starvation
•Code multilivello con feedback:
-i processi possono cambiare coda
-risolve lo starvation introducendo l'aging



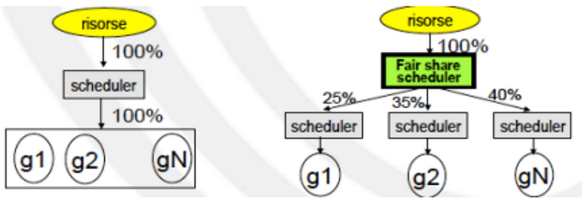
ESEMPIO CODE MULTI-LIVELLO CON FEEDBACK

•Abbiamo 3 code:
-Q_0: RR con quanto 8 ms
-Q_1: RR con quanto 16 ms
-Q_2: FCFS
•Un job entra in Q_0. Se non finisce entro il proprio quanto, viene spostato in Q_1
•Un job entra in Q_1. Se non finisce entro il proprio quanto, viene spostato in Q_2



SCHEDULING FAIR SHARE

•Le politiche di scheduling di cui abbiamo parlato fino ad ora sono orientate ai singoli processi
•Fair share cerca di dividere equamente le risorse della CPU tra le varie applicazioni (gruppi di processi)
-ogni gruppo ottiene una giusta percentuale



VALUTARE UN ALGORITMO DI SCHEDULING

Ogni sistema richiede prestazioni diverse, quindi è opportuno applicare delle tecniche per valutare gli algoritmi di scheduling in contesti diversi:
•**Modello deterministico:** viene definito un preciso carico di lavoro (workflow) e viene testato su carta con gli algoritmi
-accurato solo per il workflow specifico
•**Modello a reti di code:** si usano formule matematiche per determinare tempi di arrivo, di attesa, thoughput medio, ecc.

-sistema di calcolo descritto come una rete di server, ognuno con la propria coda
-produce risultati poco realistici

- **Simulazione:** si simula lo scheduler con dei software appositi
  -molto precisa
  -molto lenta e costosa
- **Implementazione:** si implementa l'algoritmo in un SO e si misurano le prestazioni
  -miglior metodo di valutazione