Lezione 6

Schedulazione dei processi

Sistemi operativi

17 aprile 2012

Marco Cesati

System Programming Research Group
Università degli Studi di Roma Tor Vergata

Di cosa parliamo in questa lezione?

La schedulazione dei processi

1. Schedulazione e caratteristiche dei processi
2. Algoritmi di schedulazione
3. Schedulazione nei sistemi multiprocessori
4. Linux

Schedulazione dei processi

Marco Cesati

La schedulazione dei processi

Nei SO multiprogrammati un importante componente è lo *schedulatore a breve termine* (o *scheduler*) dei processi

Lo scheduler seleziona tra tutti i processi immediatamente eseguibili su una CPU quello che deve essere effettivamente posto in esecuzione

Le caratteristiche dello scheduler influenzano in modo significativo le prestazioni dell'intero sistema di calcolo

Schema di funzionamento:

- Un processo A in esecuzione termina, blocca o sospende
- Lo scheduler seleziona un nuovo processo B da eseguire
- Viene effettuato il cambio di contesto: $A \Longrightarrow B$

Nei SO con *thread kernel* viene schedulato il singolo thread in alternativa oppure in aggiunta al singolo processo

Fasi cicliche di un processo

La frequenza di funzionamento dei circuiti della CPU è molto più elevata di quella dei circuiti di memoria

Perciò l'attività di un generico processo è costituita da una alternanza ciclica di due fasi:

1. *CPU burst*: il processo compie una sequenza di operazioni sui dati posti in registri della CPU od in memoria primaria
2. *I/O burst*: il processo è sospeso in attesa che dati necessari per continuare l'esecuzione siano trasferiti dalla memoria secondaria

Processo	CPU burst	I/O burst
CPU bound	lunghi	brevi
I/O bound	brevi	lunghi

I SO multiprogrammati sono vantaggiosi perché lo scheduler può sostituire il processo in esecuzione sospeso per un I/O burst con un altro con dati sono immediatamente disponibili

Invocazione dello scheduler

Lo scheduler deve essere invocato ogni volta che si deve effettuare un cambio di contesto:

1. quando il processo in esecuzione termina
2. quando il processo in esecuzione esegue una chiamata di sistema *bloccante* per
   - richiedere un trasferimento da memoria secondaria
   - richiedere l'attesa di un evento
   - rilasciare volontariamente la CPU
3. quando il processo in esecuzione viene posto nello stato pronto dal SO (ossia il SO lo rimuove dalla CPU)
4. quando un processo non in esecuzione diventa eseguibile

SO nonpreemptive

Un SO è *nonpreemptive* (*cooperativo*, *senza diritto di prelazione*) se lo scheduler è invocato solo quando il processo in esecuzione termina od esegue una chiamata di sistema bloccante; altrimenti è *preemptive* (*con diritto di prelazione*)

Sistemi operativi preemptive o cooperativi

Un SO cooperativo:

- Ha una organizzazione più semplice
- Un programma può monopolizzare la CPU bloccando l'intero sistema
- Utilizzato in MS Windows 3 e nei primi SO Apple

Un SO preemptive:

- Ha una organizzazione interna complessa
- Un processo in esecuzione può essere rimosso dalla CPU
  - sempre quando il processo esegue in User Mode
  - mai se il processo esegue in Kernel Mode ed il SO è *kernel nonpreemptible*
  - *quasi* sempre se il processo esegue in Kernel Mode ed il SO è *kernel preemptible*
  - Linux, Windows NT, Mac OS X hanno kernel preemptible

Metriche per gli algoritmi di schedulazione

Esistono diversi tipi di algoritmi di schedulazione

Ciascuno di essi, a parità di condizioni, può dar luogo ad un differente ordine d'esecuzione dei processi attivi nel sistema

Per valutare la qualità relativa di diversi algoritmi è utile considerare alcune misure quantitative:

- Misure relative alla CPU (da massimizzare):
  - Utilizzo della CPU
  - Produttività
- Misure relative ai tempi dei processi (da minimizzare):
  - Tempo di completamento
  - Tempo di attesa
  - Tempo di risposta

Misure relative alla CPU

Utilizzo della CPU

La frazione di tempo in un intervallo in cui la CPU esegue processi

Produttività (*throughput*)

Il numero di processi completati nell'unità di tempo

In un intervallo di tempo di due minuti vengono eseguiti e completati 50 processi, ciascuno con un tempo di esecuzione pari a 2 secondi. Quanto valgono utilizzo della CPU e produttività?

- L'utilizzo della CPU è pari a $\frac{50 \times 2}{120} = 83.\overline{3}\%$
- Il throughput è pari a $\frac{50}{120} = 41.\overline{6}\%$

La produttività dipende dai tempi di esecuzione dei processi

Misure relative ai processi

Tempo di completamento (*turnaround time*)

Intervallo tra creazione e terminazione di un processo

Tempo di attesa (*waiting time*)

Somma degli intervalli di tempo in cui un processo è rimasto nello stato di pronto (eseguibile ma non in esecuzione)

Tempo di risposta (*response time*)

Intervallo che intercorre tra la creazione di un processo e l'istante in cui esso comincia a fornire un risultato in output

Ciascuna misura si applica al singolo processo: cosa si potrebbe cercare di minimizzare?

- La media dei valori $\Rightarrow$ riduzione del valore atteso
- Il valore massimo $\Rightarrow$ riduzione del caso peggiore
- La varianza $\Rightarrow$ aumento della predicibilità

Algoritmi a priorità

- Lo scheduler del SO seleziona il processo da porre in esecuzione sulla base di un ben determinato algoritmo
- Tutti gli algoritmi di schedulazione utilizzati in pratica possono essere formulati in termini di *priorità* dei processi
  - Lo scheduler seleziona sempre un processo nello stato pronto avente il valore di priorità più alto
  - Se due processi hanno la stessa priorità la scelta dipende dall'algoritmo
  - In pratica, alcuni SO associano ad un valore numerico più piccolo una priorità più alta, per altri SO vale l'opposto

Priorità statica e dinamica

- Priorità statica: costante assegnata in fase di creazione
- Priorità dinamica: può cambiare durante l'esecuzione

Priorità esterna ed interna

- Priorità esterna: assegnata dall'utente o amministratore
- Priorità interna: assegnata dal kernel del SO

Problemi degli algoritmi a priorità

Inversione di priorità incontrollata

Il fenomeno per cui un processo continua l'esecuzione per un tempo indefinito pur se un processo di priorità superiore ad esso è nello stato di pronto

- L'inversione di priorità incontrollata è inevitabile quando lo scheduler è di tipo cooperativo (nonpreemptive)
- Con uno scheduler di tipo preemptive, la priorità di un processo che passa nello stato pronto viene confrontata con quella del processo in esecuzione e, se necessario, si effettua un cambio di contesto

Attesa indefinita (*starvation*)

Un processo con priorità molto bassa rimane indefinitivamente in stato di pronto a causa di processi di priorità superiore

- Se la priorità è dinamica si può innalzare la priorità dei processi nello stato di pronto (invecchiamento o aging)

Algoritmo FCFS (First Come, First Served)

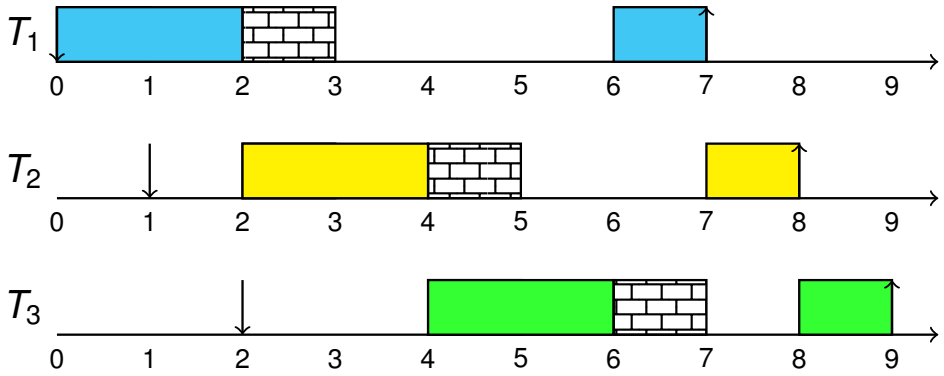
Schedulazione in ordine di arrivo

La priorità è proporzionale al momento in cui il processo è posto nello stato di pronto

- È un algoritmo a priorità dinamica e interna
- È di tipo collaborativo (nonpreemptive)
- Implementato tramite una struttura di dati FIFO
  - Lista di PCB che include i processi pronti
  - I processi posti in stato di pronto aggiunti in coda alla lista
  - Lo scheduler seleziona il processo in testa alla lista
- È facile da implementare ma ha parecchi svantaggi:
  - Tempo d'attesa medio lungo
  - Non adatto a sistemi con time sharing
  - Con un processo CPU bound e molti processi I/O bound si crea un *effetto convoglio* che porta a sotto-utilizzare la CPU

Esempio di schedulazione FCFS

T_1: creato a 0, durata totale 3, oper. di I/O dopo 2 lunga 1
T_2: creato a 1, durata totale 3, oper. di I/O dopo 2 lunga 1
T_3: creato a 2, durata totale 3, oper. di I/O dopo 2 lunga 1



Coda | 1 | 2 | 2 3 | 3 1 | 3 1 | 1 2 | 1 2 | 2 3 | 3 | |

	T_1	T_2	T_3	media	max
Tempo di completamento	7	7	7	7	7
Tempo di attesa	3	3	3	3	3

Algoritmo SJF (Shortest Job First)

Schedulazione per brevità

La priorità è proporzionale alla durata del successivo *CPU burst* (sequenza di operazioni prima di una operazione di I/O)

- È un algoritmo a priorità dinamica e interna
- Più è breve la sequenza di operazioni di CPU da eseguire prima di una operazione di I/O, maggiore è la priorità
- Priorità identiche sono gestite tramite FCFS
- Può essere in alternativa di tipo:
  - cooperativo, oppure
  - preemptive (SRTF, Shortest Remaining Time First)
- È un algoritmo ottimale: minimizza il tempo d'attesa medio
  - anticipando un processo con una sequenza più breve diminuisco il suo tempo d'attesa più di quanto aumenti il tempo d'attesa del processo posticipato
- Svantaggio: è richiesta la lunghezza del *CPU burst*
  - Non adatto per schedulatori a breve termine
  - Più adatto per schedulatori a lungo termine

Predizione tramite media esponenziale

Difficoltà maggiore nell'implementazione dell'algoritmo SJF: impossibile calcolare la lunghezza del CPU burst

Gli scheduler a lungo termine possono utilizzare come lunghezza un tempo limite d'esecuzione fornito dall'utente

Uno scheduler a breve termine deve predire la durata del CPU burst di un processo osservando la durata dei precedenti:

Media esponenziale

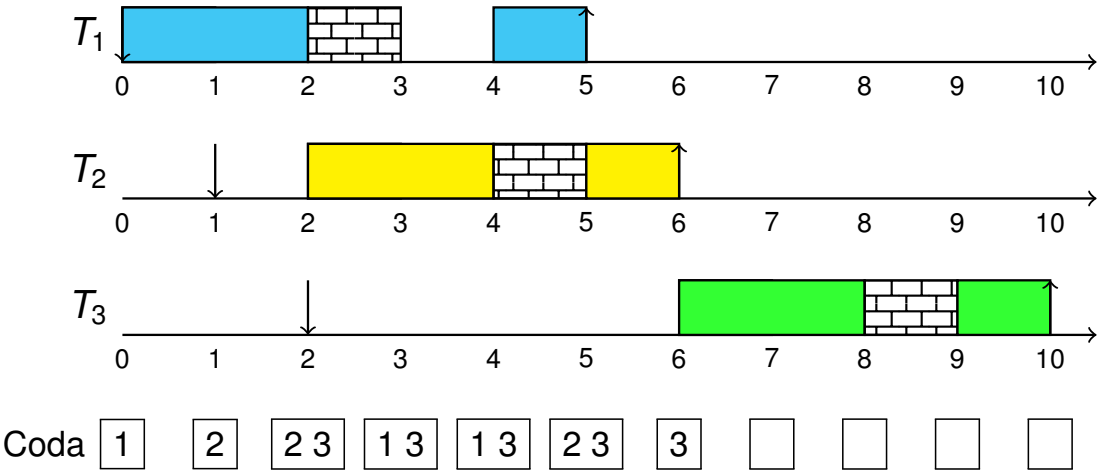
Se t è l'ultima osservazione fatta e τ_0 è la media precedente, la nuova media esponenziale è

$$\tau = \alpha \cdot t + (1 - \alpha) \cdot \tau_0$$

ove $\alpha \in [0, 1]$ è la relazione tra nuova e vecchie osservazioni

Esempio di schedulazione SJF (cooperativo o preemptive)

T_1: creato a 0, durata totale 3, oper. di I/O dopo 2 lunga 1
T_2: creato a 1, durata totale 3, oper. di I/O dopo 2 lunga 1
T_3: creato a 2, durata totale 3, oper. di I/O dopo 2 lunga 1



	T_1	T_2	T_3	media	max
Tempo di completamento	5	5	8	6	8
Tempo di attesa	1	1	4	2	4

Algoritmo RR (Round Robin)

Schedulazione circolare

La priorità è proporzionale al momento in cui il processo è posto nello stato di pronto, ma viene annullata se il processo viene eseguito per un tempo predefinito

- È un algoritmo a priorità dinamica e interna
- Tipica implementazione: usa due liste di processi ordinate per priorità
  - 1^a lista per processi con tempo d'esecuzione non esaurito
  - 2^a lista per processi con tempo d'esecuzione esaurito
- Lo scheduler preleva il processo da eseguire dalla testa della 1^a lista
- Un processo in esecuzione che esaurisce il proprio quanto viene tolto dalla CPU e accodato nella 2^a lista
- Un processo che diventa pronto viene accodato nella 1^a lista (il tempo ancora a disposizione non cambia)
- Quando la 1^a lista si svuota, il ruolo delle liste viene invertito, e a tutti i processi è assegnato un nuovo quanto
- L'algoritmo è simile a FCFS con diritto di prelazione

Quanto di tempo

Quanto di tempo (*time slice*)

Il tempo massimo d'esecuzione consentito ad un processo prima che venga prelazionato

La durata del quanto di tempo incide in modo cruciale sulle prestazioni dell'algoritmo RR

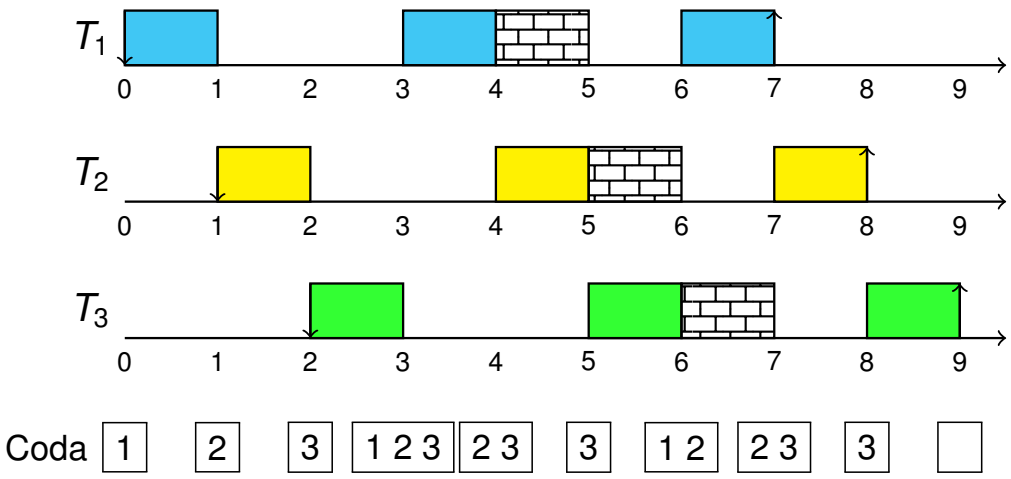
- Un quanto molto lungo rende le prestazioni di RR simili a quelle di FCFS
- Un quanto molto breve comporta molte prelazioni
  - Di conseguenza si verificano molti cambi di contesto
- Tipicamente:
  - un cambio di contesto costa circa 10 μsec
  - un quanto di tempo è lungo tra i 10 msec e i 200 msec

Epoca

L'intervallo di tempo necessario perché tutti i processi nel sistema esauriscano il quanto di tempo

Esempio di schedulazione RR con quanto unitario

T_1: creato a 0, durata totale 3, oper. di I/O dopo 2 lunga 1
T_2: creato a 1, durata totale 3, oper. di I/O dopo 2 lunga 1
T_3: creato a 2, durata totale 3, oper. di I/O dopo 2 lunga 1



	T_1	T_2	T_3	media	max
Tempo di completamento	7	7	7	7	7
Tempo di attesa	3	3	3	3	3

Algoritmo MLQS (MultiLevel Queue Scheduling)

Gli algoritmi visti finora permettono di schedulare una classe omogenea di processi

Per schedulare processi con caratteristiche differenti (non omogenei) è necessario utilizzare algoritmi gerarchici

Schedulazione a code multiple

I vari processi sono assegnati a code differenti, in base alle loro caratteristiche; ciascuna classe di processi è gestita con un algoritmo di schedulazione *ad hoc*

Ad esempio, un SO può distinguere tra processi interattivi e processi in background:

- I processi interattivi sono gestiti tramite uno scheduler RR
- I processi in background tramite uno scheduler FCFS

Algoritmo MLQS (MultiLevel Queue Scheduling) (2)

Oltre agli scheduler di basso livello è necessario realizzare un algoritmo di schedulazione per le varie classi di processi

Ad esempio si potrebbe avere, in alternativa:

- Un algoritmo preemptive a priorità statica
  - ogni classe di processi ha precedenza assoluta sulle classi di processi di priorità inferiore
- Un algoritmo Round Robin non preemptive con quanti di tempo differenti per ogni classe
  - Ciascuna classe di processi ha garantita una frazione minima del tempo di CPU

L'assegnazione di un nuovo processo ad una classe è una scelta definitiva effettuata

- dall'utente o amministratore di sistema (facile)
- dallo scheduler MLQS, in base a determinate caratteristiche del nuovo processo (difficile)

Algoritmo MLFQS (MultiLevel Feedback Queue Scheduling)

Schedulazione a code multiple con retroazione

Basato sull'algoritmo di schedulazione a code multiple (MLQS), con in più la possibilità di cambiare la classe di appartenenza dei processi in base al loro comportamento

Ad esempio: se un SO definisce due classi di processi, interattivi e non interattivi, lo scheduler può classificare un processo in esecuzione osservando le sue operazioni di I/O

Lo scheduler MLFQS è un algoritmo generale che deve essere realizzato definendo:

- il numero e la caratterizzazione delle classi di processo
- l'algoritmo di schedulazione di ciascuna classe
- l'algoritmo di schedulazione generale delle classi
- l'algoritmo per determinare quando spostare un processo da una classe all'altra

Sistemi multiprocessori

Sistema multiprocessore

Sistema di calcolo integrante diverse unità di calcolo indipendenti che condividono il sistema di memoria e le periferiche di I/O

Processore multi-core

Circuito integrato contenente due o più *core* di computazione, ciascuno dei quali equivalente ad una CPU tradizionale

I core nello stesso processore possono condividere alcune risorse hardware, quali ad esempio:

- Memoria statica (ad es., cache di secondo livello)
- Memory Management Unit (MMU)

Perché sono stati introdotti i processori multicore?

La scala di integrazione dei circuiti integrati continua a crescere, consentendo di ridurre la dimensione dei circuiti, ma non si può più aumentare la loro frequenza di funzionamento

Sistemi multiprocessori (2)

Processore multithread (o con hyperthread)

Un processore provvisto di vari insiemi di registri associati a diversi flussi di esecuzione; i flussi condividono i blocchi funzionali del processore

Perché sono stati introdotti i processori multithread?

Quando un flusso di esecuzione si blocca in attesa di un trasferimento di dati dalla memoria centrale, un altro flusso può eseguire calcoli su dati già disponibili

Esistono quindi diversi livelli di parallelismo hardware:

- Processori fisici
- Core multipli entro un processore fisico
- Thread d'esecuzione entro un core

Per il SO le varie forme di parallelismo sono in una certa misura equivalenti (*processore virtuale*)

Affinità dei processi

- I processori moderni, fino al livello di core, utilizzano memorie statiche veloci per aumentare le prestazioni (cache, TLB)
- Questi meccanismi tendono a “legare” un processo già eseguito al processore su cui è stato eseguito per ultimo
  - Eseguendo sullo stesso processore esiste una certa probabilità di trovare in cache dati letti in precedenza
  - Eseguendo su processore differente ogni dato deve essere recuperato dalla memoria centrale
- I SO tendono perciò a gestire i multiprocessori tramite uno scheduler gerarchico:
  - Per ciascun processore si definisce un elenco di processi legati ad esso, che viene schedulato separatamente
  - Uno scheduler di livello superiore assegna ad ogni nuovo processo un determinato processore
  - L’utente o amministratore di sistema generalmente può legare un processo ad uno specifico insieme di processori (*processor affinity*)

Bilanciamento del carico

Lo scheduler gerarchico per sistemi multiprocessore potrebbe portare ad una situazione patologica:

- Un processore con la coda piena esegue in continuazione, mentre un altro processore con la coda vuota è inattivo
- Di conseguenza il SO non sfrutta un processore

Tutti i SO con scheduler gerarchico integrano meccanismi per effettuare il bilanciamento del carico

- Meccanismo di *push migration*: periodicamente il SO controlla lo stato delle code dei processori ed eventualmente migra un processo da una coda piena ad una con meno elementi
- Meccanismo di *pull migration*: quando uno scheduler locale determina che la propria coda di processi è vuota, esamina le code degli altri processori ed eventualmente effettua una migrazione

Vantaggi e svantaggi dei due tipi di meccanismi?

La push migration è più costosa, ma tendenzialmente assicura migliore equità nella gestione dei processi

Schedulazione dei processi
Marco Cesati

Schema della lezione
Schedulazione
Algoritmi
Multiprocessori
Linux

Esempio: lo scheduler CFS di Linux

Lo scheduler di Linux CFS (Completely Fair Scheduler) è stato introdotto nella versione 2.6.23 (ottobre 2007)

- Implementa l'idea di *classi di schedulazione* modulari
  - È possibile introdurre nuovi politiche e algoritmi di schedulazione senza dover riscrivere l'intero algoritmo
- Overhead costante all'aumentare dei processi ("$O(1)$")
- Scheduler multiprocessore
  - con affinità dei processori
  - bilanciamento del carico
  - basato sul riconoscimento di cache condivise e hyperthreading

Esempio: lo scheduler CFS di Linux (2)

Implementa un algoritmo MLQS con le seguenti classi di processi:

- real time, con 99 livelli di priorità statica esterna
  - variante FCFS (`SCHED_FIFO`)
  - variante Round Robin (`SCHED_RR`), con quanto di tempo impostabile per ciascun processo
- normali (`SCHED_NORMAL`)
  - algoritmo "fair" in cui la priorità è proporzionale a quanto poco è stato eseguito il processo
  - non basato su code ma su alberi bilanciati *Red-Black*
  - possibile per l'utente variare la priorità relativa dei task
- batch (`SCHED_BATCH`), per i processi non interattivi
  - Simili ai processi normali, ma con meno prelazioni quando in esecuzione in User Mode
- idle (`SCHED_IDLE`), per processi di priorità molto bassa

La percentuale di CPU dedicata alle varie classi è configurabile