Sistemi Operativi
06 luglio 2010
Compitino 1

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (a) Che cosa significa che il kernel è prelazionabile? Che cosa sono i punti di prelazionabilità?
   (b) In quali situazioni è importante che il kernel sia prelazionabile?
   (c) Elencare vantaggi e svantaggi di un kernel prelazionabile.

   Risposta:

   1. Un kernel si dice prelazionabile quando consente la prelazione di un processo anche quando operi in kernel-mode (e.g., quando esegue il codice di una chiamata di sistema). I punti di prelazionabilità individuano delle posizioni nel codice delle chiamate di sistema in cui è possibile che il processo venga prelazionato: solitamente i punti di prelazionabilità vengono individuati in posizioni sicure del kernel, ovvero, laddove non vi siano in corso operazioni di modifica delle strutture dati di quest'ultimo (per non rischiare di creare delle inconsistenze).
   2. Nei sistemi operativi per applicazioni tradizionali il kernel non è prelazionabile in quanto non vi sono esigenze di risposte in tempi così rapidi come quelli tipici di questi sistemi. Invece nei sistemi real-time avere un kernel prelazionabile è fondamentale per evitare che una chiamata di sistema troppo lunga porti al fallimento dei vincoli temporali dei vari task.
   3. I vantaggi di un kernel prelazionabile consistono nell'avere dei tempi di risposta molto più rapidi, raggiungendo un grado di parallelismo molto più alto rispetto ai kernel tradizionali. Gli svantaggi sono rappresentati dalla maggior complessità del kernel stesso e dal rischio di deadlock/inconsistenze in caso di errori di progettazione dei punti di prelazione o dell'utilizzo delle primitive di sincronizzazione.

2. (a) In quali situazioni può essere attivato lo scheduling della CPU?
   (b) Quando un algoritmo di scheduling è preemptive? Quali sono i vantaggi e gli svantaggi di un algoritmo preemptive?
   (c) Gli scheduler Round Robin mantengono una lista di tutti i processi pronti, in cui ogni processo compare una sola volta. Che cosa accade se un processo apparisse due volte nella lista? Ci sono situazioni in cui questo potrebbe essere permesso?

   Risposta:

   1. Lo scheduling della CPU può essere attivato nelle seguenti circostanze:
      (a) un processo viene creato ed entra nella cosa dei pronti;
      (b) un processo passa dallo stato di esecuzione allo stato di attesa;
      (c) un processo passa dallo stato di esecuzione allo stato pronto;
      (d) un processo passa dallo stato di attesa allo stato pronto;
      (e) un processo termina.
   2. Un algoritmo di scheduling si dice preemptive se può interrompere l'esecuzione di un processo a favore di un altro processo e può quindi essere attivato ogni volta che un processo passa nella coda dei pronti (coda ready), oltre che ovviamente anche in altri casi. I vantaggi di un algoritmo preemptive sono essenzialmente dei tempi di risposta migliori e la garanzia che nessun processo riesca a monopolizzare la CPU senza rilasciarla. Gli svantaggi sono relativi alla condivisione dei dati fra processi; infatti, se un processo sta manipolando dei dati utilizzati anche da altri processi e viene prelazionato c'è il rischio che questi rimangano in uno stato inconsistente e generino così degli errori. Per evitare tutto ciò è necessario un attento uso di primitive come mutex e semafori per garantire un accesso esclusivo e corretto alle risorse condivise.
   3. Inserendo due puntatori allo stesso processo, quest'ultimo, nel caso in cui ad esempio i due puntatori siano consecutivi, si vedrebbe assegnare un tempo di CPU doppio rispetto agli altri processi. Infatti, scaduto il primo quanto di tempo, il primo puntatore al PCB verrebbe spostato in fondo alla coda, ma il secondo puntatore conferirebbe un ulteriore quanto di tempo al processo in questione. In generale quindi il processo beneficerebbe di un tempo di CPU doppio nel corso di ogni singola scansione della coda dei processi pronti. Un vantaggio derivante dall'utilizzo di questo sistema è quello di poter assegnare quanti di tempo diversi a seconda delle necessità di ogni singolo processo. Uno svantaggio potrebbe essere il rallentamento di alcuni processi a

causa dell'inserimento "non accorto" dei puntatori "doppi" nella coda. Un ulteriore aspetto negativo è costituito dal fatto che nel caso di puntatori consecutivi allo stesso PCB ci sarebbe comunque uno spreco di tempo per l'esecuzione delle system call dovute agli interrupt del timer di sistema per gestire l'assegnazione dei quanti di tempo (tali chiamate di sistema potrebbero essere evitate in quanto il processo che va in esecuzione rimane sempre lo stesso).

3. Si consideri un sistema con scheduling SJF con prelazione (cioè SRTF), ove $\alpha = 0,5$ e $\tau_0 = 30$ msec. All'istante 0 il processore si libera e tre processi, P_1, P_2, P_3, sono in coda ready. Finora i processi P_1, P_2 sono andati in esecuzione due volte con CPU burst $25, 40$ msec per P_1 e $30, 20$ msec per P_2; mentre P_3 è andato in esecuzione una volta con CPU burst di 60 msec.
   Si determini:
   (a) Quale processo viene selezionato dallo scheduler all'istante 0?
   (b) All'istante 10 msec entra nella coda ready un nuovo processo P_4 con CPU burst previsto di 20 msec. Il processo selezionato precedentemente è ancora in esecuzione. Che cosa succede?
   (c) Che cosa succede quando il processo in esecuzione termina il suo burst?

   Risposta:

   1. Per P_1 abbiamo $\tau_1 = \frac{1}{2} \cdot 30 + \frac{1}{2} \cdot 25 = 15 + 12,5 = 27,5$, $\tau_2 = \frac{1}{2} \cdot 27,5 + \frac{1}{2} \cdot 40 = 33,75$, per P_2 abbiamo $\tau_1 = \frac{1}{2} \cdot 30 + \frac{1}{2} \cdot 30 = 30$, $\tau_2 = \frac{1}{2} \cdot 30 + \frac{1}{2} \cdot 20 = 15 + 10 = 25$ ed infine per P_3 abbiamo $\tau_1 = \frac{1}{2} \cdot 30 + \frac{1}{2} \cdot 60 = 45$. Quindi all'istante 0 SRTF sceglie P_2.
   2. All'istante 10 msec quando entra nella coda ready un nuovo processo P_4 con CPU burst previsto di 20 msec, continua P_2 perché gli mancano meno di 20 ms ($25 - 10 = 15$ ms) che è il burst previsto per P_4.
   3. Quando P_2 termina, va in esecuzione P_4.

4. (a) Cosa si intende per *race condition* (corsa critica)? Il verificarsi di corse critiche rappresenta un evento positivo o negativo per un sistema di calcolo?
   (b) Cosa si intende per *busy wait* (attesa attiva)? Si citi un aspetto positivo ed uno negativo di questa tecnica.
   (c) Descrivere brevemente il costrutto *monitor*.

   Risposta:

   (a) Si parla di race condition quando più processi accedono concorrentemente agli stessi dati e il risultato dipende dall'ordine di interleaving dei processi. Il verificarsi di corse critiche rappresenta un evento negativo per un sistema di calcolo in quanto introduce non determinismo e possibili inconsistenze nelle strutture dati condivise.
   (b) Per *busy wait* (attesa attiva) si intende l'attendere il verificarsi di un certo evento "sprecando" tempo di CPU per controllare ciclicamente se sia avvenuto o meno (ad esempio controllando il valore di una variabile). È un meccanismo semplice da implementare, ma può portare a consumi inaccettabili di CPU; pertanto sarebbe da utilizzare solo in caso di attese molto brevi).
   (c) Un monitor è un tipo di dato astratto che fornisce funzionalità di mutua esclusione: sostanzialmente è unaa collezione di dati privati e funzioni/procedure per accedervi. I processi possono chiamare le procedure, ma non accedere alle variabili locali. Un solo processo alla volta può eseguire codice di un monitor. Il programmatore raccoglie quindi i dati condivisi e tutte le sezioni critiche relative in un monitor: questo risolve il problema della mutua esclusione. Vengono implementati dal compilatore con dei costrutti per mutua esclusione (p.e.: inserendo automaticamente `lock_mutex` e `unlock_mutex` all'inizio ed alla fine di ogni procedura).

5. Si consideri la seguente situazione, dove P_0, P_1, P_2 sono tre processi in esecuzione, C è la matrice delle risorse correntemente allocate, Max è la matrice del numero massimo di risorse assegnabili ad ogni processo e A è il vettore delle risorse disponibili:

	C			Max		
	A	B	C	A	B	C
P_0	1	2	0	1	5	1
P_1	0	1	0	2	5	2
P_2	2	3	0	2	4	2

Sistemi Operativi
06 luglio 2010
Compitino 1

Available (A)		
A	B	C
1	5	x

(a) Calcolare la matrice R delle richieste.

(b) Determinare il minimo valore di x tale che il sistema si trovi in uno stato sicuro.

Risposta:

(a) La matrice R delle richieste è data dalla differenza $Max - C$:

	R	
A	B	C
0	3	1
2	4	2
0	1	2

(b) Se $x = 0$, allora non esiste nessuna riga R_i tale che $R_i \leq A$; quindi il sistema si trova in uno stato di deadlock. Se $x = 1$, allora l'unica riga di R minore o uguale a A è la prima. Quindi possiamo eseguire P_0 che, una volta terminato, restituisce le risorse ad esso allocate aggiornando A al valore $(2, 7, 1)$. A questo punto non esiste alcuna riga di R minore o uguale al vettore A e quindi il sistema è in stato di deadlock.

Il valore minimo di x per cui lo stato risulta sicuro è 2; infatti in questo caso esiste la sequenza sicura $\langle P_0, P_1, P_2 \rangle$. Dapprima si esegue P_0, generando il valore $(2, 7, 2)$ di A, poi si esegue P_1 portando A al valore $(2, 8, 2)$. A questo punto si conclude la sequenza eseguendo P_2 e generando il valore finale di A, ovvero, $(4, 11, 2)$.