Sistemi Operativi
5 dicembre 2006
Compitino A

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Che cos'è la multiprogrammazione? Si può realizzare su un sistema monoprocessore?

   Risposta: La nozione di multiprogrammazione prevede la possibilità di mantenere in memoria più processi contemporaneamente in modo da mantenere la CPU occupata il maggior tempo possibile. Infatti, non appena un processo richiede al sistema operativo di eseguire un'operazione di I/O, quest'ultimo può dedicare l'utilizzo della CPU ad un altro processo, invece di attendere passivamente il completamento della suddetta operazione. Questa tecnica è realizzabile anche su un sistema monoprocessore.

2. (a) Si dia la definizione di thread e si descriva come i thread sono rappresentati nei sistemi operativi.
   (b) Si diano due esempi di situazioni in cui i sistemi a thread sono vantaggiosi rispetto ai processi tradizionali.

   Risposta:

   (a) I thread sono unità di esecuzione all'interno di uno stesso processo; ogni thread è caratterizzato da un proprio PC, da un proprio insieme dei valori dei registri, da un proprio stack ed un proprio stato di esecuzione, mentre tutte le risorse rimanenti (spazio in memoria, file aperti ecc.) vengono condivise fra thread appartenenti allo stesso processo. Vi sono due possibili implementazioni dei thread: a livello utente ed a livello kernel. Nel primo caso ogni processo mantiene una tabella dei thread propria in cui viene tenuta traccia di tutti i thread del processo (PC, valori dei registri, stack, stato di esecuzione). Nel caso invece di un'implementazione a livello kernel viene tenuta, a fianco della tabella dei processi, una tabella dei thread nello spazio di memoria del nucleo del sistema operativo (quindi c'è un'unica tabella per tutti i thread in esecuzione nel sistema). Ovviamente nel primo caso si ha un'efficienza maggiore che nel secondo in quanto tutte le operazioni sui thread vengono eseguite in user space, per mezzo di un'apposita libreria, mentre nel secondo caso bisogna ricorrere ad opportune chiamate di sistema passando ogni volta da user mode a kernel mode.

   (b) Un esempio di situazione in cui sono vantaggiosi i thread, rispetto ai processi tradizionali, è nel caso di un word processor, in cui un thread si può occupare della gestione della GUI (e quindi dell'interazione con l'utente), mentre un altro thread può eseguire delle operazioni in background come, ad esempio, il salvataggio o la stampa del documento. In questo modo, mentre il documento viene salvato/stampato, l'utente può proseguire ad interagire con il word processor; senza l'utilizzo di thread separati invece l'utente sarebbe costretto ad attendere il completamento dell'operazione di I/O.

   Un altro esempio dei vantaggi dei thread rispetto ai processi tradizionali si ha in programmi di tipo server come i web server in cui ogni richiesta viene presa in carico da un thread separato. Nel caso in cui si gestiscano le richieste con processi separati infatti si possono avere dei seri problemi di performance e di "spreco" di risorse all'aumentare del numero di richieste. Ciò non si verifica utilizzando i thread in quanto le operazioni di creazione di nuovi thread sono molto più veloci e meno onerose rispetto a quelle di creazione di nuovi processi.

3. Si consideri un sistema con scheduling della CPU a priorità con tre code, A, B, C, di priorità decrescente, con prelazione tra code. Le code A e B sono round robin con quanto di 10 e 15 ms, rispettivamente; la coda C è FCFS. Se un processo nella coda A o B consuma il suo quanto di tempo, viene spostato in fondo alla coda B o C, rispettivamente.

   Nelle code A, B, C entrano i seguenti processi:

	coda	arrivo	burst
P_1	A	0	30ms
P_2	C	5	20ms
P_3	B	15	15ms
P_4	A	20	15ms

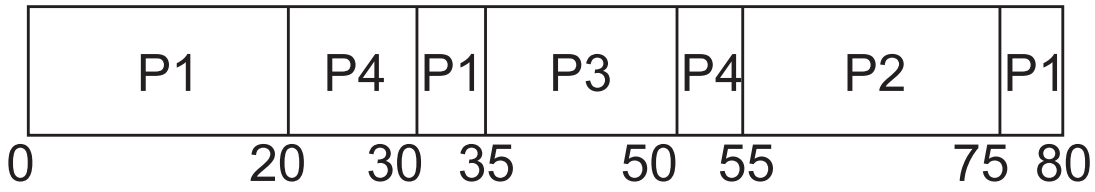
   Si determini:

   (a) il diagramma di GANTT relativo all'esecuzione dei quattro processi;

   (b) il tempo di attesa medio;

(c) il tempo di reazione medio.

Risposta:



(a)

(b) tempo di attesa medio=$\frac{(30-20+75-35)+(55-5)+(35-15)+(50-30)}{4} = \frac{50+50+20+20}{4} = 35$ ms;

(c) tempo di reazione medio=$\frac{0+(55-5)+(35-15)+0}{4} = \frac{50+20}{4} = 17,5$ ms.

4. 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	0	3	1	0	5	1
P_1	2	2	2	2	4	3
P_2	1	3	0	2	4	1

Available (A)		
A	B	C
1	2	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) Matrice delle richieste $R = Max - C$:

	R	
A	B	C
0	2	0
0	2	1
1	1	1

(b) Il minimo valore di x tale da rendere lo stato sicuro è 0. Infatti $R_0 \leq A = (1, 2, 0)$ e quindi si può eseguire P_0 fino alla sua terminazione. A questo punto A diventa $A + C_0 = (1, 2, 0) + (0, 3, 1) = (1, 5, 1)$ e quindi $R_1 \leq A$. Dopo la terminazione di P_1, A diventa $A + C_1 = (1, 5, 1) + (2, 2, 2) = (3, 7, 3)$. Quindi è possibile mandare in esecuzione anche P_2 (dato che $R_2 \leq A$) ed ottenere il valore finale di A, ovvero, $A + C_2 = (3, 7, 3) + (1, 3, 0) = (4, 10, 3)$. Ne segue che la sequenza sicura è $\langle P_0, P_1, P_2 \rangle$.

5. Si elenchino (spiegandole) le quattro condizioni di Coffman. Sono condizioni necessarie o sufficienti perché si possa verificare un deadlock?

Risposta: Le quattro condizioni di Coffman sono:

1. Mutua esclusione: ogni risorsa è assegnata ad un solo processo, oppure è disponibile.
2. Hold&Wait: i processi che hanno richiesto ed ottenuto delle risorse, ne possono richiedere altre.
3. Mancanza di prerilascio: le risorse che un processo detiene possono essere rilasciate dal processo solo volontariamente.
4. Catena di attesa circolare di processi: esiste un sottoinsieme di processi $\{P_0, P_1, ..., P_n\}$ tali che P_i è in attesa di una risorsa che è assegnata a $P_{i+1 \bmod n}$.

Le condizioni sono necessarie, ma non sufficienti per il verificarsi di uno stallo.

6. Qual è la ragione principale che impedisce di adottare l'algoritmo del banchiere in un sistema operativo per eludere i deadlock?

   Risposta: La ragione principale che impedisce di adottare l'algoritmo del banchiere in un sistema operativo per eludere i deadlock è che necessita di conoscere a priori il tipo e la quantità di risorse che ogni processo andrà ad utilizzare nel corso della sua esecuzione. Inoltre, l'algoritmo si basa su un numero di processi fissato a priori, mentre nella realtà nuovi processi vengono continuamente creati dinamicamente.

7. I monitor ed i semafori sono costrutti applicabili anche a sistemi distribuiti (ovvero, che non condividono memoria)? Motivare la risposta e, in caso di risposta negativa, descrivere un meccanismo alternativo adatto a tale scopo.

   Risposta: No, monitor e semafori necessitano di una forma di memoria condivisa e quindi non sono costrutti applicabili a sistemi distribuiti. Per questi ultimi si può utilizzare il modello dello scambio di messaggi in cui si utilizzano due primitive (chiamate di sistema o funzioni di libreria):

   - `send(destinazione, messaggio)`: spedisce un messaggio ad una certa destinazione; solitamente non bloccante.
   - `receive(sorgente, &messaggio)`: riceve un messaggio da una sorgente; solitamente bloccante (fino a che il messaggio non è disponibile).

   Tali funzioni non necessitano di nessuna forma di condivisione di memoria e quindi sono applicabili anche per mettere in comunicazione dei sistemi distribuiti.