Sistemi Operativi
11 giugno 2012
Compitino 1

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Un sistema multiprogrammato utilizza un grado di multiprogrammazione molto grande.Viene proposto di raddoppiare il troughput del sistema migliorando/sostituendo le sue componenti hardware. Il risultato desiderato verrebbe raggiunto con qualcuna delle seguenti proposte?
   (a) Sostituire la CPU con una che abbia velocità doppia rispetto alla prima.
   (b) Raddoppiare la quantità di memoria.
   (c) Sostituire la CPU con una che abbia velocità doppia rispetto alla prima e raddoppiare la quantità di memoria.

   Risposta:
   (a) (2 punti) Sostituire la CPU con una che abbia velocità doppia rispetto alla prima non cambia il grado di multiprogrammazione. Quindi, ci potrebbero essere lunghi periodi di tempo in cui la CPU non esegue alcun processo e, di conseguenza, il throughput sarebbe limitato dalla disponibilità della memoria.
   (b) (2 punti) Raddoppiare la quantità di memoria, aumenta il grado di multiprogrammazione (dato che possono essere eseguiti più processi), ma il throughput sarebbe limitato dalla velocità della CPU e quindi potrebbe non raddoppiare.
   (c) (2 punti) Sostituire la CPU con una che abbia velocità doppia rispetto alla prima e raddoppiare la quantità di memoria è la soluzione migliore a patto di caricare in quest'ultima un insieme adeguato di processi.

2. Quale delle seguenti transizioni di stato per un processo può causare la transizione di stato *waiting* → *ready* per uno o più degli altri processi?
   (a) Un processo avvia un'operazione di I/O e passa nello stato *waiting*.
   (b) Un processo termina.
   (c) Un processo effettua la richiesta per una risorsa e passa nello stato *waiting*.
   (d) Un processo invia un messaggio.
   (e) Un processo effettua una transizione di stato *waiting* → *waiting swapped*.

   Risposta:
   (a) (1 punto) No, non influenza il passaggio da *waiting* a *ready* di un altro processo.
   (b) (1 punto) Sì se, per esempio, si tratta della terminazione di un processo figlio di cui il padre era in attesa o se un altro processo era in attesa di una risorsa detenuta dal processo terminato.
   (c) (1 punto) No, non influenza il passaggio da *waiting* a *ready* di un altro processo.
   (d) (1 punto) Sì: un altro processo poteva essere in attesa di ricevere un messaggio.
   (e) (1 punto) No, lo swapping su disco di un processo in stato *waiting* non può influenzare il passaggio da *waiting* a *ready* di un altro processo.

3. Discutere lo speed-up dell'elaborazione delle seguenti applicazioni in un sistema che ha (i) una CPU e (ii) più di una CPU.
   (a) In un server che gestisce molte richieste a un tasso molto elevato vengono creati molti thread, dove il servizio di una richiesta coinvolge sia attività di CPU che di I/O.
   (b) L'elaborazione di un'espressione $z := a * b + c * d$ viene effettuata creando due processi figli che valutano $a * b$ e $c * d$.

   Risposta:
   (a) (2 punti) L'unico modo di ottenere uno speed-up dell'elaborazione, sia nel caso di una CPU che nel caso multiprocessore, è quello di sovrapporre l'attività di CPU di un thread con quella di I/O di un altro thread.
   (b) (2 punti) In questo caso con una sola CPU non è possibile avere alcuno speed-up dato che i due processi richiedono entrambi soltanto attività di CPU. Inoltre si perderebbe molto tempo a creare e sincronizzare i processi, diminuendo di fatto la velocità di elaborazione. Nel caso multiprocessore tale attività di creazione e gestione dei processi renderebbe ancora difficile ottenere uno speed-up a causa del sovraccarico di gestione.

Sistemi Operativi
11 giugno 2012
Compitino 1

4. Classificare ognuna delle seguenti affermazioni come vera o falsa.
   (a) Un'applicazione può contenere una race condition solo se il sistema su cui è in esecuzione l'applicazione possiede più di una CPU.
   (b) Un processo può subire una starvation all'entrata di una sezione critica se l'implementazione della sezione critica non soddisfa la condizione di attesa limitata.
   (c) Un processo può subire una starvation all'entrata di una sezione critica se l'implementazione della sezione critica non soddisfa la condizione di progresso.

   Risposta:
   (a) (2 punti) Falsa: un'applicazione può essere costituita da più thread; quindi si possono verificare delle race condition indipendentemente dal numero di CPU.
   (b) (2 punti) Vera: può non entrare in un tempo limitato perché scavalcato da altri processi.
   (c) (2 punti) Vera: qualche processo che sta eseguendo al di fuori della sua sezione critica potrebbe impedirgli l'accesso.

5. Un processo in background dovrebbe funzionare in modo da non degradare significativamente il servizio fornito agli altri processi. Quale delle seguenti alternative si suggerisce per implementarlo?
   (a) Assegnare la minima priorità al processo di background.
   (b) Attribuire un quanto minore al processo di background rispetto agli altri processi.

   Risposta:
   (a) (2 punti) Assegnare la minima priorità potrebbe provocare una starvation del processo in background.
   (b) (2 punti) Attribuire al processo in background un quanto di tempo minore è la soluzione corretta, dato che, anche se per un tempo più breve, gli si garantirà di essere regolarmente eseguito.

6. Il nuovo stato di allocazione di un sistema dopo aver accettato una richiesta di risorsa non è uno stato di allocazione sicuro secondo l'algoritmo del banchiere.
   (a) Ciò implica che si verificherà sicuramente un deadlock?
   (b) Il sistema riesce ad effettuare una transizione in uno stato di allocazione sicuro? Se sì, dare un esempio che mostri tale transizione.

   Risposta:
   (a) (2 punti) No: potrebbe non verificarsi un deadlock nel caso in cui qualche processo rilasciasse un numero sufficiente di risorse.
   (b) (2 punti) Consideriamo due processi P_1 e P_2 nella seguente situazione:

Richieste massime			*Risorse allocate*		
	R_1	R_2		R_1	R_2
P_1	3	3	P_1	0	1
P_2	3	2	P_2	1	1

   Risorse massime: $(3, 3)$
   Risorse disponibili: $(2, 1)$

   Lo stato corrente è sicuro, perché P_2 può procedere. Invece se P_1 chiede ed ottiene una risorsa (ad esempio R_2), allora il nuovo stato non sarà più sicuro. Tuttavia, se in seguito, P_1 rilascerà la risorsa in oggetto, allora si ritornerà in uno stato sicuro (in cui P_2 potrà chiedere le risorse e terminare).