Sistemi Operativi
5 dicembre 2006
Compitino B

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (a) Si descriva il meccanismo attraverso cui i programmi richiamano i servizi del sistema operativo. Si faccia qualche esempio.

   (b) Qual è la differenza tra un interrupt e una trap?

   Risposta:

   (a) I programmi richiamano i servizi del sistema operativo per mezzo delle chiamate di sistema (system call): sono solitamente disponibili come speciali istruzioni assembler o come delle funzioni nei linguaggi che supportano direttamente la programmazione di sistema (ad esempio, il C). Esistono vari tipi di chiamate di sistema relative al controllo di processi, alla gestione dei file e dei dispositivi, all'ottenimento di informazioni di sistema, alle comunicazioni. L'invocazione di una chiamata di sistema serve ad ottenere un servizio dal sistema operativo; ciò viene fatto passando dal cosiddetto *user mode* al *kernel mode* per mezzo dell'istruzione speciale `TRAP`. Infatti, il codice relativo ai servizi del sistema operativo è eseguibile soltanto in kernel mode per ragioni di sicurezza. Una volta terminato il compito relativo alla particolare chiamata di sistema invocata, il controllo ritorna al processo chiamante passando dal kernel mode allo user mode.

   (b) Un interrupt è un segnale inviato dal controller di un dispositivo alla CPU per segnalare un evento (ad esempio un errore od il completamento di un'operazione), mentre una trap è un'interruzione software generata da un processo per mezzo di un'apposita istruzione.

2. Si dia la definizione di processo e si descriva come i processi sono rappresentati nei sistemi operativi.

   Risposta: Un processo è un programma in esecuzione nel sistema; quindi non consiste del solo codice, ma anche di tutte le risorse correlate: il program counter (PC), i registri, lo stack, lo stato di esecuzione, lo spazio di indirizzamento, le variabili globali, i file aperti, i timer in scadenza, i segnali e le routine di gestione dei segnali, le informazioni di accounting ecc. Nei sistemi operativi si tiene traccia dei processi in esecuzione in un'apposita struttura dati del kernel, ovvero, la tabella dei processi in cui ogni entry è costituita da un PCB (Process Control Block) in cui sono memorizzate tutte le informazioni sul processo relativo.

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

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

	coda	arrivo	burst
P_1	A	0	25ms
P_2	C	5	20ms
P_3	B	15	15ms
P_4	A	20	15ms

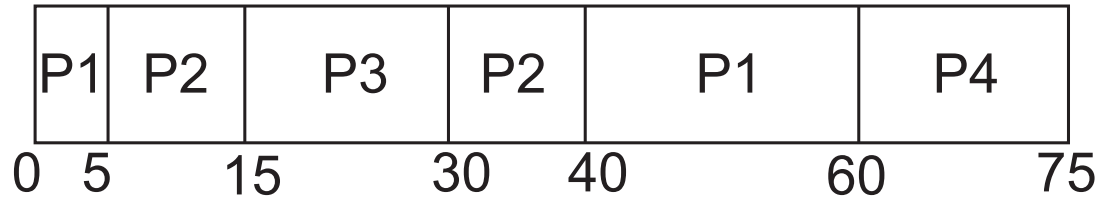
   Si determini:

   (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=$\frac{(40-5)+(30-15)+0+(60-20)}{4} = \frac{35+15+40}{4} = 22,5$ ms;

(c) tempo di reazione medio=$\frac{0+0+0+(60-20)}{4} = \frac{40}{4} = 10$ 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
x	4	1

(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 $R_0 \leq A = (0, 4, 1)$ e quindi si può eseguire P_0 fino alla sua terminazione. A questo punto A diventa $A + C_0 = (0, 4, 1) + (0, 3, 1) = (0, 7, 2)$ e quindi $R_1 \leq A$. Dopo la terminazione di P_1, A diventa $A + C_1 = (0, 7, 2) + (2, 2, 2) = (2, 9, 4)$. Quindi è possibile mandare in esecuzione anche P_2 (dato che $R_2 \leq A$) ed ottenere il valore finale di A, ovvero, $A + C_2 = (2, 9, 4) + (1, 3, 0) = (3, 12, 4)$. Ne segue che la sequenza sicura è $\langle P_0, P_1, P_2 \rangle$.

5. Si illustrino le condizioni per una buona soluzione del problema della sezione critica.

Risposta: Le condizioni per una buona soluzione del problema della sezione critica sono:

- Mutua esclusione: se il processo P_i sta eseguendo la sua sezione critica, allora nessun altro processo può eseguire la propria sezione critica.
- Progresso: nessun processo in esecuzione fuori dalla sua sezione critica può bloccare processi che desiderano entrare nella propria sezione critica.
- Attesa limitata: se un processo P ha richiesto di entrare nella propria sezione critica, allora il numero di volte che si concede agli altri processi di accedere alla propria sezione critica prima del processo P deve essere limitato.

Di solito si suppone che ogni processo venga eseguito ad una velocità non nulla, mentre non si suppone niente sulla velocità relativa dei processi (e quindi sul numero e tipo di CPU).

6. Quando uno stato di un sistema si può definire sicuro (safe)?

Risposta: Uno stato di un sistema si dice sicuro (safe) se non è presente un deadlock ed esiste una sequenza di esecuzione sicura, ovvero, è possibile stabilire un ordine di esecuzione dei processi in modo che tutti possano giungere alla terminazione anche richiedendo il numero massimo di risorse immediatamente.

7. Che cosa si intende per *istruzione TSL*? In cosa può risultare utile?

Risposta: Per istruzione TSL si intende Test-and-Set-Lock, ovvero, un'istruzione che controlla e modifica il contenuto di una parola atomicamente. Lo pseudo codice relativo è il seguente:

```
function Test-and-Set (var target: boolean): boolean;
  begin
    Test-and-Set := target;
    target := true;
  end;
```

Questi due passi devono essere implementati in modo atomico in assembler (ovvero, il bus della memoria deve essere bloccato mentre vengono eseguite le due operazioni in modo da impedire l'accesso ad altri processi).

Dato che l'atomicità delle istruzioni TSL mette al riparo dalle race condition, questo tipo di istruzione risulta utile per implementare le zone di entrata e uscita delle sezioni critiche (anche se con attesa attiva).