Sistemi Operativi
09 luglio 2013
Compitino 1

Si risponda ai seguenti quesiti, giustificando le risposte.
Gli esercizi e le domande marcate con l'asterisco (*) devono essere svolti soltanto da chi ha in piano di studi l'esame di Sistemi Operativi da 9 o 12 CFU.

1. (*) Si considerino un sistema multiprocessore e un programma multithread scritto con il modello da molti a molti. Si ipotizzi un numero più alto di thread a livello utente nel programma rispetto al numero di processori nel sistema. Si analizzi che cosa implica, in termini di efficienza, ciascuna delle seguenti possibilità.
   (a) Il numero di thread del kernel assegnati al programma è minore del numero di processori.
   (b) I thread del kernel assegnati al programma sono in numero uguale al numero dei processori.
   (c) Il numero di thread del kernel assegnati al programma è maggiore del numero di processori, ma minore del numero di thread a livello utente.

   Risposta:
   (a) Essendoci sicuramente dei thread a livello kernel che saranno associati a più thread a livello utente, ci saranno più context-switch a livello utente (meno costosi in quanto non richiedono un cambio di modalità e possono essere gestiti in user space) che a livello kernel, ma il sistema sarà sottoutilizzato nel caso in cui non ci sia un numero sufficiente di processi da saturare l'utilizzo dei processori. Inoltre il processo in questione non potrà trarre il massimo beneficio in termini di parallelismo, rispetto al numero di processori del sistema.
   (b) Se i thread del kernel assegnati al programma sono in numero uguale al numero dei processori, nella situazione ideale in cui ogni thread del kernel sia associato ad un processore diverso e non vi siano altri processi a contendere l'utilizzo delle CPU, ci saranno soltanto context-switch a livello utente (infatti avremo ancora dei thread a livello kernel associati a diversi thread a livello utente). Il sistema sarà quindi utilizzato al massimo anche nel caso in cui il processo relativo al programma in questione sia l'unico in esecuzione nel sistema.
   (c) In quest'ultimo caso ci saranno ancora diversi thread a livello utente associati a singoli thread a livello kernel ed in più vi saranno parecchi context-switch anche a livello kernel, dato che non tutti i thread riusciranno a vedersi assegnato un processore. Il sistema quindi sarà sovraccaricato.

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?

   Risposta:
   (a) Lo scheduling della CPU può essere attivato nelle seguenti circostanze:
      1. un processo viene creato ed entra nella cosa dei pronti;
      2. un processo passa dallo stato di esecuzione allo stato di attesa;
      3. un processo passa dallo stato di esecuzione allo stato pronto;
      4. un processo passa dallo stato di attesa allo stato pronto;
      5. un processo termina.

   (b) 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. I processi P_1, P_2, P_3, di priorità decrescente, con richieste di CPU totali di 8, 12, 8 s rispettivamente, sono in coda ready. L'algoritmo di scheduling della CPU è basato su priorità con prelazione.

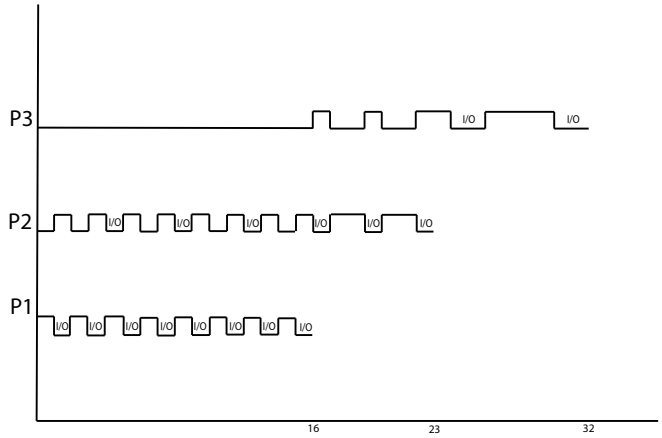
Sistemi Operativi
09 luglio 2013
Compitino 1

Ogni processo P_i ha un comportamento ciclico: in ogni ciclo consuma c_i secondi di CPU ed esegue operazioni di I/O per o_i secondi, secondo la seguente tabella

	c_i	o_i	c_{tot}
P_1	1	1	8
P_2	2	1	12
P_3	4	2	8

L'algoritmo di scheduling viene eseguito ad intervalli di 1 secondo. Si calcolino i tempi di turnaround per i processi.

Risposta: Come si deduce dal diagramma seguente, i tempi di turnaround di P_1, P_2 e P_3 sono, rispettivamente, 16s, 23s e 32s.



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) Descrivere due problemi che possono derivare dall'attesa attiva di un evento (busy wait).

   (c) I monitor ed i semafori sono costrutti applicabili anche a sistemi distribuiti? Motivare la risposta e in caso di risposta negativa descrivere un meccanismo alternativo adatto a tale scopo.

   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) Un aspetto negativo dell'attesa attiva è il consumo "inutile" del tempo di CPU per controllare il valore della variabile. Un altro problema è dato dal fatto che l'attesa attiva di un processo a bassa priorità può ritardare/bloccare l'esecuzione di processi a maggiore priorità in attesa della stessa risorsa (inversione di priorità).

   (c) 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.

5. Si supponga che in una struttura vi sia un solo bagno e che sia necessario disciplinarne l'accesso tenendo presente quanto segue:

   - non deve mai capitare che nel bagno siano presenti contemporaneamente uomini e donne;

- un uomo (risp. una donna) prima di entrare deve controllare che nel bagno non siano presenti delle donne (risp. degli uomini): se la condizione non è verificata deve attendere.

Si completi la specifica del seguente monitor in modo da soddisfare i requisiti menzionati.

```
monitor Bathroom
    ...
    integer nmen, nwomen;

    procedure manEnter();
        begin
            while(nwomen>0) do ...
            ...
        end;

    procedure manExit();
        begin
            nmen := nmen-1;
            if(...) then ...
        end;

    procedure womanEnter();
        begin
          ...
          ...
        end;

    procedure womanExit();
        begin
          ...
          ...
        end;

    nmen := 0;
    nwomen := 0;
end monitor;
```

Risposta:

```
monitor Bathroom
    condition bathroomFree
    integer nmen, nwomen;

    procedure manEnter();
        begin
            while(nwomen>0) do wait(bathroomFree);
            nmen := nmen + 1;
        end;

    procedure manExit();
        begin
            nmen := nmen-1;
            if(nmen=0) then signal(bathroomFree);
        end;

    procedure womanEnter();
        begin
            while(nmen>0) do wait(bathroomFree);
```

```
            nwomen := nwomen + 1;
        end;

    procedure womanExit();
        begin
            nwomen := nwomen-1;
            if(nwomen=0) then signal(bathroomFree);
        end;

    nmen := 0;
    nwomen := 0;
end monitor;
```

Punteggi:

- 9/12 CFU: 1a (3pt.), 1b (3 pt.), 1c (3 pt.), 2a (2 pt.), 2b (2 pt.), 3 (5 pt.), 4a (3 pt.), 4b (3 pt.), 4c (3 pt.), 5 (4 pt.)
- 6 CFU: 2a (3 pt.), 2b (3 pt.), 3 (6 pt.), 4a (4 pt.), 4b (4 pt.), 4c (4 pt.), 5 (6 pt.)