

Sistemi Operativi

04 luglio 2012

Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Classificare ognuna delle seguenti affermazioni come vera o falsa.
 - (a) Per ottenere un maggior throughput un SO multiprogrammato assegna priorità maggiore ai programmi CPU-bound.
 - (b) Se un kernel multiprogrammato scopre che l'efficienza della CPU è bassa, dovrebbe rimuovere dalla memoria un programma I/O bound.
 - (c) Se il quanto di tempo in un sistema time-sharing è troppo lungo, i processi termineranno la loro esecuzione nello stesso ordine in cui erano stati avviati.
 - (d) Due persone che utilizzano lo stesso time-sharing allo stesso momento potrebbero ricevere tempi di risposta molto differenti.

Risposta:

- (a) (1,5 punti) Vero a condizione che i processi CPU-bound non richiedano tempi di esecuzione mediamente superiori a quelli I/O-bound. Infatti il throughput è dato dal numero di processi terminati nell'unità di tempo e questo valore non è detto che aumenti se si favoriscono processi CPU-bound molto più lunghi a terminare di brevi processi I/O-bound (conteggiando ovviamente anche i tempi richiesti per le operazioni di I/O).
- (b) (1,5 punti) Falso: in condizioni normali (ovvero, memoria non in esaurimento) rimuovere un processo I/O-bound non aumenta l'efficienza della CPU (in quanto i processi I/O-bound pur eseguendo per la maggior parte del tempo attività di I/O, impegnano minimamente anche la CPU). Piuttosto sarebbe più opportuno far partire l'esecuzione di un ulteriore processo.
- (c) (1,5 punti) Vero: se il quanto di tempo in un sistema time-sharing è troppo lungo, i processi avranno il tempo di terminare completamente la propria esecuzione nel corso del primo quanto di tempo assegnato, facendo degenerare lo scheduling in un FCFS.
- (d) (1,5 punti) Vero: il tempo di risposta può essere influenzato da diversi fattori, come, ad esempio, la natura dei processi lanciati in esecuzione dalle due persone. Ad esempio, il lancio da parte di un utente di un processo "sfavorito" in quanto CPU-bound potrebbe riscontrare un tempo di risposta superiore rispetto al lancio da parte di un altro utente di un processo interattivo.

2. Un SO supporta sia thread utente che thread kernel. Si discutano le seguenti affermazioni.
 - (a) Se l'elaborazione da eseguire in un thread è CPU-bound, conviene creare un thread kernel se il sistema dispone di più CPU; in caso contrario conviene creare un thread utente.
 - (b) Se l'elaborazione da eseguire in un thread è I/O-bound, conviene creare un thread utente se il processo che lo contiene non ha un thread kernel; in caso contrario conviene creare un thread kernel.

Risposta:

- (a) (2 punti) Un thread kernel garantisce un maggior parallelismo, ma anche un maggior overhead. Quindi, se il sistema dispone di più CPU, la creazione di thread kernel consente di sfruttare il parallelismo fornito dal sistema, altrimenti è meglio creare thread utente per evitare di sovraccaricare inutilmente il sistema.
- (b) (2 punti) Se il processo in cui viene eseguito il thread I/O-bound contiene anche un'attività CPU-bound, allora conviene creare un thread kernel per l'attività I/O bound in modo da sfruttare il parallelismo del sistema. In assenza di attività CPU-bound nel processo, creare thread kernel per l'attività I/O-bound produrrebbe soltanto del sovraccarico, senza nessun beneficio.

3. I processi periodici P_1, P_2, P_3 hanno tempi di servizio (CPU-burst) di $5ms, 3ms, 10ms$, rispettivamente. I periodi di P_1 e P_2 sono $T_1 = 25ms$ e $T_2 = 8ms$. Qual è il valore minimo di T_3 per cui la politica di scheduling RMS (Rate Monotonic Scheduling) rispetterebbe le deadline di tutti i processi?

Risposta: (4 punti) Una soluzione si può ottenere risolvendo la seguente disequazione:

$$\sum_{i=1}^m \frac{C_i}{T_i} \leq 1$$

Sistemi Operativi

04 luglio 2012

Compito

Quindi nel nostro caso:

$$\frac{5}{25} + \frac{3}{8} + \frac{10}{T_3} \leq 1$$

Ne segue che deve essere $8 \cdot T_3 + 15 \cdot T_3 + 400 \leq 40 \cdot T_3$, ovvero, $17 \cdot T_3 \geq 400$, ovvero, $T_3 \geq 23,53$ ms.

Un'altra stima può essere calcolata considerando la disequazione $\sum_{i=1}^m \frac{C_i}{T_i} \leq m(2^{1/m} - 1)$. Nel nostro caso, risolvendola, rispetto a T_3 , otteniamo $T_3 \cdot [120 \cdot (2^{\frac{1}{3}} - 1) - 23] \geq 400$, ovvero $T_3 \geq \frac{400}{120 \cdot 0,26 - 23} = 48,78$ ms.

4. Un ponte che collega due sponde di un fiume è così stretto da permettere il passaggio delle auto su un'unica corsia a senso alternato (quindi le macchine possono muoversi concorrentemente sul ponte solo se vanno nella stessa direzione). Per semplicità, ci si riferisce alle macchine che procedono in una direzione con l'espressione "RedCars" e a quelle che procedono nella direzione opposta con l'espressione "BlueCars". Si scriva il codice di accesso al ponte, usando un monitor, in modo che:

1. in ogni istante il ponte viene utilizzato da veicoli che transitano in una sola direzione;
2. se ci sono veicoli in attesa di transitare in entrambe le direzioni, questi *si devono alternare* nel passaggio sul ponte;
3. se non ci sono veicoli in attesa in una direzione, allora un qualsiasi numero di veicoli provenienti dalla direzione opposta può attraversare il ponte.

Si noti che le regole sopra specificate (in particolare la regola 2) differiscono da quelle considerate nell'esempio visto a lezione.

Risposta: (5 punti) Una possibile soluzione che utilizza i monitor è la seguente:

```

monitor FairBridge
    condition bridgeFree;
    integer nred, nblue, waitred, waitblue;
    boolean blueturn;

    procedure redEnter();
    begin
        waitred := waitred+1;
        while(nblue>0 or (waitblue>0 and blueturn))
            do wait(bridgeFree);
        waitred := waitred-1;
        nred := nred +1;
        blueturn := true;
    end;

    procedure redExit();
    begin
        nred := nred-1;
        if(nred=0) then signal(bridgeFree);
    end;

    procedure blueEnter();
    begin
        waitblue := waitblue+1;
        while(nred>0 or (waitred>0 and not(blueturn)))
            do wait(bridgeFree);
        waitblue := waitblue-1;
        nblue := nblue +1;
        blueturn := false;
    end;

    procedure blueExit();

```

Sistemi Operativi

04 luglio 2012

Compito

```

begin
    nblue := nblue-1;
    if(nblue=0) then signal(bridgeFree);
end;

nred := 0;
nblue := 0;
waitred := 0;
waitblue := 0;
blueturn := true;
end monitor;

```

In questo modo ogni “RedCar”, accedendo al ponte, imposta il turno per le “BlueCar” (`blueturn := true`) e, viceversa, ogni “BlueCar”, accedendo al ponte, imposta il turno per le “RedCar” (`blueturn := false`). Un’auto, prima di accedere al ponte, controlla che non ci siano macchine che marciano nella direzione opposta sul ponte e che non ci siano macchine in attesa con il turno di passaggio a loro favorevole. L’implementazione della primitiva `signal` in questo caso deve essere tale da risvegliare tutti i thread/processi in attesa sulla condition variable `bridgeFree`. In questo modo tutti i thread che erano in attesa possono ricontrillare (uno per volta) la condizione a guardia del `while` ed accedere al ponte o tornare a sospendersi in attesa di un’altra segnalazione dell’evento `bridgeFree`. Quindi tutte le auto che erano in coda ai due estremi del ponte passeranno alternandosi nelle due direzioni.

5. Si consideri il seguente array bidimensionale: `int X[64][64];`

Si supponga che un sistema abbia 4 frame e ciascun frame sia di 128 parole (un intero occupa una parola). I programmi che manipolano l’array X stanno esattamente dentro una pagina e occupano sempre la pagina 0. Negli altri 3 frame viene fatto lo swapping dei dati in ingresso e uscita. L’array X è memorizzato in ordine di riga più grande (cioè nella memoria $X[0][1]$ segue $X[0][0]$). Quale dei seguenti frammenti di codice genererà il minor numero di errori di pagina? Perché? Si calcoli il numero totale di errori di pagina nei due casi.

Frammento A

```

for (int j = 0; j < 64; j++)
    for (int i = 0; i < 64; i++) X[i][j] = 0;

```

Frammento B

```

for (int i = 0; i < 64; i++)
    for (int j = 0; j < 64; j++) X[i][j] = 0;

```

Risposta: (5 punti) Nel caso del frammento A, l’accesso agli elementi dell’array X avviene nell’ordine seguente:

	stessa pag.		
$X[0,0]$,	$X[1,0]$,	...,	$X[63,0]$
$X[0,1]$,	$X[1,1]$,	...,	$X[63,1]$
...			
$X[0,63], X[1,63], \dots, X[63,63]$			

Quindi dovremo accedere a 32 pagine logiche diverse per ogni riga, ovvero, avremo $32 \times 64 = 2048$ errori di pagina.

Invece, nel caso del frammento B, l’accesso agli elementi dell’array X avviene nell’ordine seguente:

$X[0,0], X[0,1], \dots, X[0,63]$	}	stessa pag.
$X[1,0], X[1,1], \dots, X[1,63]$	}	stessa pag.
...		
$X[63,0], X[63,1], \dots, X[63,63]$		

Quindi dovremo accedere ad una pagina logica diversa ogni 2 righe dell’array, ovvero, avremo 32 errori di pagina.

Sistemi Operativi

04 luglio 2012

Compito

6. Si illustri la differenza fra dispositivi a blocchi ed a carattere, indicando le possibili modalità di accesso (e.g., sequenziale, ad accesso casuale) disponibili per ognuna delle due categorie. Si facciano inoltre due esempi di periferiche classificabili come dispositivi a blocchi e due esempi di periferiche classificabili come dispositivi a carattere.

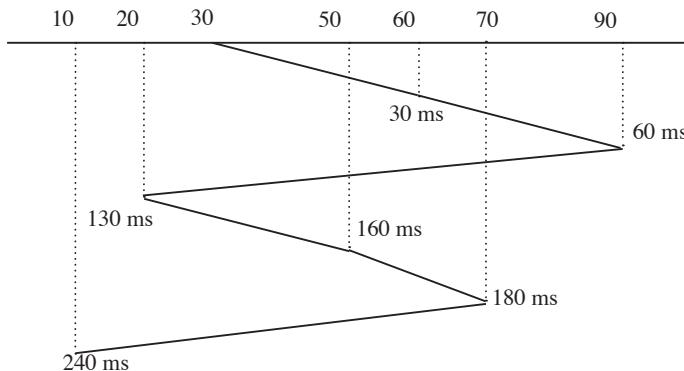
Risposta: (3 punti) I dispositivi a blocchi permettono l'accesso diretto ad un insieme finito di blocchi di dimensione costante. L'unità delle operazioni di trasferimento (input/output) è quindi il blocco. Il tipico esempio di questo tipo di dispositivi è rappresentato dai dischi. I dispositivi a carattere invece generano o accettano uno stream di dati (formato da singoli byte/caratteri) non strutturati su cui non permettono indirizzamento. I tipici rappresentanti di questa categoria di dispositivi sono tastiera, mouse, unità a nastro. Quindi i dispositivi a blocchi supportano sia l'accesso sequenziale che diretto (random), mentre i dispositivi a carattere supportano solamente l'accesso sequenziale.

7. Si consideri un disco gestito con politica C-LOOK (LOOK circolare) con direzione di servizio verso tracce con numero crescente. Inizialmente la testina è posizionata sul cilindro 30; lo spostamento ad una traccia adiacente richiede 1 ms. Al driver di tale disco arrivano richieste per i cilindri 90, 20, 60, 70, 50, 10, rispettivamente agli istanti 0 ms, 10 ms, 20 ms, 80 ms, 90 ms, 150 ms. Si trascuri il tempo di latenza.

1. In quale ordine vengono servite le richieste?
2. Il tempo di attesa di una richiesta è il tempo che intercorre dal momento in cui è sottoposta al driver a quando viene effettivamente servita. Qual è il tempo di attesa medio per le sei richieste in oggetto?

Risposta:

1. (3 punti) Le richieste vengono servite nell'ordine 60, 90, 20, 50, 70, 10:



2. (1 punto) Il tempo di attesa medio per le sei richieste in oggetto è

$$\frac{(60-0)+(130-10)+(30-20)+(180-80)+(160-90)+(240-150)}{6} = \frac{60+120+10+100+70+90}{6} = \frac{450}{6} = 75 \text{ ms.}$$