Sistemi Operativi
11 luglio 2008
Esame completo

Si risponda ai seguenti quesiti, giustificando le risposte.

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.

    1. Il numero di thread del kernel assegnati al programma è minore del numero di processori.
    2. I thread del kernel assegnati al programma sono in numero uguale al numero dei processori.
    3. 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:

    1. 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.
    2. 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.
    3. 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. Data una variante dell'algoritmo di scheduling RR in cui gli elementi della coda dei processi pronti sono puntatori ai PCB:

    1. si descriva l'effetto dell'inserimento di due puntatori allo stesso processo nella coda dei processi pronti;
    2. si descrivano i principali vantaggi e svantaggi di questo schema;
    3. si inventi una modifica all'algoritmo RR ordinario che consenta di ottenere lo stesso effetto senza duplicare i puntatori.

    Risposta:

    1. Inserendo due puntatori allo stesso processo, quest'ultimo, nel caso in cui ad esempio i due puntatori siano consecutivi, si vedrebbe assegnare un tempo di CPU doppio rispetto agli altri processi. Infatti, scaduto il primo quanto di tempo, il primo puntatore al PCB verrebbe spostato in fondo alla coda, ma il secondo puntatore conferirebbe un ulteriore quanto di tempo al processo in questione. In generale quindi il processo beneficerebbe di un tempo di CPU doppio nel corso di ogni singola scansione della coda dei processi pronti.
    2. Un vantaggio derivante dall'utilizzo di questo sistema è quello di poter assegnare quanti di tempo diversi a seconda delle necessità di ogni singolo processo. Uno svantaggio potrebbe essere il rallentamento di alcuni processi a causa dell'inserimento "non accorto" dei puntatori "doppi" nella coda. Un ulteriore aspetto negativo è costituito dal fatto che nel caso di puntatori consecutivi allo stesso PCB ci sarebbe comunque uno spreco di tempo per l'esecuzione delle system call dovute agli interrupt del timer di sistema per gestire l'assegnazione dei quanti di tempo (tali chiamate di sistema potrebbero essere evitate in quanto il processo che va in esecuzione rimane semopre lo stesso).
    3. Per ottenere lo stesso effetto basterebbe assegnare un quanto di tempo "personalizzato" in base alle esigenze di ogni singolo processo.

3. 1. Si confrontino gli algoritmi di scheduling della CPU a priorità statica e quelli a priorità dinamica.
   2. Può un processo a priorità bassa influire sull'esecuzione di un processo a più alta priorità? Se sì, si diano alcuni esempi.
   3. Il problema descritto sopra può verificarsi con thread di livello utente?
   4. Può verificarsi se l'algoritmo di scheduling è RR?

   Risposta:

   1. Gli algoritmi di scheduling della CPU a priorità statica assegnano delle priorità ai processi che vengono mantenute inalterate per tutto il corso della loro esecuzione. Gli algoritmi a priorità dinamica invece prevedono dei meccanismi di aggiornamento delle priorità assegnate inizialmente ai processi in base al "comportamento" di questi ultimi durante la propria esecuzione. In tal modo, per esempio, possono venire favoriti i processi interattivi che vedranno crescere la loro priorità a dispetto dei processi CPU-bound. In generale, per evitare dei fenomeni di starvation, gli algoritmi di scheduling della CPU a priorità dinamica, prevedono anche dei meccanismi di aging, per aumentare progressivamente la priorità dei processi che da più tempo sono in attesa di ottenere la CPU (in modo da evitare che non riescano mai ad essere eseguiti).
   2. Sì, un processo a bassa priorità può influire sull'esecuzione di un processo a più alta priorità grazie al fenomeno noto come "inversione di priorità". In questo caso infatti se una data risorsa viene assegnata ad un processo a bassa priorità e quest'ultimo non la rilascia in tempo, può succedere che venga sospeso in favore di un processo a maggiore priorità. Se anche quest'ultimo dovesse necessitare della medesima risorsa per proseguire la propria esecuzione, si troverebbe ad essere bloccato dal processo a minore priorità detentore della risorsa contesa. In questo modo, pur vedendosi assegnare la CPU, non potrebbe proseguire, mentre il processo a bassa priorità non riuscirebbe mai ad andare in esecuzione, completando il proprio task e liberando la risorsa contesa.
   3. Sì, il fenomeno descritto al punto precedente può verificarsi anche con i thread a livello utente con in più il rischio di bloccare tutti i thread utente relativi allo stesso processo.
   4. Utilizzando l'algoritmo di scheduling RR invece non si corre il rischio che avvenga il problema descritto in precedenza in quanto tale algoritmo assegna ciclicamente lo stesso quanto di tempo di CPU a tutti i processi in coda.

4. Si consideri un sistema con memoria paginata a due livelli, le cui page table siano mantenute in memoria principale. Il tempo di accesso alla memoria principale sia $t = 50ns$.

   1. Qual è il tempo effettivo di accesso alla memoria?
   2. Aggiungendo un TLB, con tempo di accesso $\epsilon = 2ns$, quale hit rate dobbiamo avere per un degrado delle prestazioni del 10% rispetto a t?

   Risposta:

   1. Il tempo effettivo di accesso alla memoria è $3t$, ovvero, 150 ns; infatti sono necessari 50 ns per accedere alla page table esterna, 50 ns per accedere alla page table interna ed infine 50 ns per accedere alla locazione nel frame fisico in memoria.
   2. Un degrado del 10% rispetto a t significa un EAT pari a $1,1 \cdot t$, ovvero, 55 ns. Quindi si ha quanto segue (α rappresenta l'hit rate):

$$\begin{aligned} EAT &= \epsilon + \alpha t + (1-\alpha)3t \\ 55 &= 2 + 50\alpha + (1-\alpha)\cdot 150 \\ 55 &= 152 - 100\alpha \end{aligned}$$

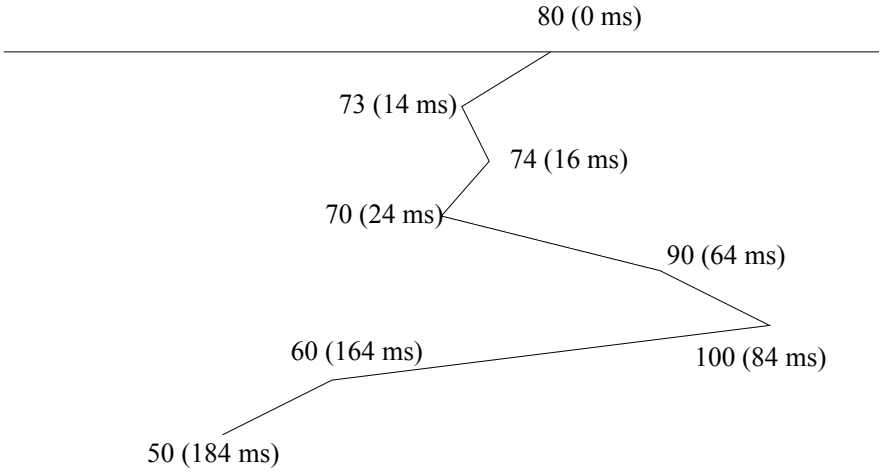
   da cui si ricava $\alpha = \frac{97}{100} = 0,97$ (97%).

5. Si consideri un disco gestito con politica SSTF. Inizialmente la testina è posizionata sul cilindro 80; lo spostamento ad una traccia adiacente richiede 2 ms. Al driver di tale disco arrivano richieste per i cilindri 70, 74, 100, 90, 50, 60 rispettivamente agli istanti 0 ms, 14 ms, 22 ms, 40 ms, 50 ms, 70 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 richieste in oggetto?

Risposta:

1. Le richieste vengono soddisfatte nell'ordine 74, 70, 90, 100, 60, 50, come evidenziato dal seguente diagramma:



2. Il tempo di attesa medio per le richieste in oggetto è: $\frac{(24-0)+(16-14)+(64-40)+(84-22)+(164-70)+(184-50)}{6} = \frac{24+2+24+62+94+134}{6} = \frac{340}{6} = 56,67ms.$

6. Suddividere il carico tra più dischi che cooperano, per offrire l'immagine di un disco unitario virtuale più efficiente, cosa comporta dal punto di vista dell'affidabilità? In particolare il Mean Time Between Failure (MTBF) del sistema dei dischi aumenta o diminuisce?

   Risposta: Mentre le prestazioni ovviamente aumentano (per via della distribuzione del carico di lavoro fra più dischi che lavorano in parallelo), l'affidabilità diminuisce. In particolare il tempo medio fra due guasti consecutivi (MTBF) diminuisce; per tale motivo si rendono necessarie architetture ridondanti come quelle descritte dai livelli RAID.

7. Si elenchino i tre princìpi su cui si può basare l'autenticazione dell'utente in un sistema operativo.

   Risposta: I tre princìpi su cui si può basare l'autenticazione dell'utente in un sistema operativo sono i seguenti:

   1. identicare qualcosa che lutente conosce (e.g., password);
   2. identicare qualcosa che lutente possiede (e.g., smartcard);
   3. identicare qualche caratteristica sica dellutente (e.g., impronte digitali).

Il punteggio attribuito ai quesiti è il seguente: 6, 6, 6, 6, 6, 3, 3. Totale 36 punti.