

Sistemi Operativi

16 febbraio 2011

Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Si descriva l'algoritmo di scheduling della CPU in Linux moderno.

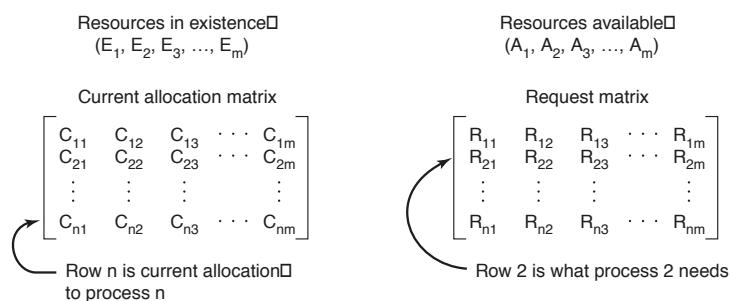
Risposta: (3 punti) L'algoritmo di scheduling di Linux moderno gira in tempo costante, $O(1)$, ed è adatto ai sistemi SMP (Symmetric MultiProcessing). Lo scheduling è con prelazione ed è basato su due classi di priorità: real time (0–99) ed altri processi (100–140). A valori numerici più bassi corrispondono priorità più alte ed a priorità più basse corrispondono quanti più lunghi (in modo da favorire i processi interattivi). Ogni processore mantiene una coda di esecuzione (runqueue) con due liste di task: attivi e scaduti. Un task è attivo se non ha terminato il suo quanto, altrimenti è scaduto; le due liste sono ordinate secondo la priorità dei task e lo scheduler sceglie il task attivo di maggior priorità. Quando l'array attivo si svuota, i due array si scambiano di ruolo. Mentre i task real-time hanno priorità statica, gli altri task hanno priorità dinamica basata su un valore nice +5 o -5, a seconda del grado di interattività: per esempio, per favorire i processi interattivi, tempi di attesa più lunghi sui dispositivi di I/O danno un bonus di -5. La priorità dinamica di un task viene ricalcolata quando passa da attivo a scaduto.

2. Si consideri un sistema su cui vengono eseguiti 10 processi con prevalenza di I/O e un processo con prevalenza di elaborazione. Supponiamo che i primi processi richiedano un'operazione di I/O ogni millisecondo di elaborazione della CPU e che ciascuna di tali operazioni sia completata in 10 millisecondi. Si assuma inoltre che il tempo necessario per il cambio di contesto sia di 0,1 millisecondi e che tutti i processi siano operazioni a lungo termine. Si calcoli l'utilizzo della CPU in presenza di scheduler RR se il quanto di tempo è pari a 10 millisecondi.

Risposta: (4 punti) Nei primi 11 ms la CPU serve i 10 processi B_1, \dots, B_{10} per 1 ms ciascuno più 1 ms totale di attività di context switch. All'istante 11 si ha 0,1 ms di context switch, poi parte il processo A per 10 ms; nel frattempo B_1, \dots, B_{10} completano la loro attività di I/O e tornano in coda ready. Dunque ripartono B_1, \dots, B_{10} e si arriva all'istante 32,1. Poi riparte A fino a 42,2 ecc. Il comportamento è ciclico con durata del ciclo di 21,1 ms. All'interno di tale ciclo la CPU è utilizzata per 20 ms, mentre 1,1 ms sono impiegati per l'attività di context switch. Quindi la percentuale di utilizzo della CPU è data da $\frac{20}{21,1} \cong 0,95$, ovvero, il 95%.

3. Si descriva l'algoritmo di rilevazione dello stallo basato sulla matrice di allocazione delle risorse, nel caso di più risorse per ogni classe.

Risposta: (3 punti) Si tratta di un algoritmo per la rilevazione dello stallo. Le strutture dati utilizzate dall'algoritmo di rilevazione dello stallo sono le seguenti, dove E_i rappresenta il numero di risorse esistenti della classe i , A_i rappresenta il numero di risorse disponibili della classe i , C_{ij} il numero di risorse della classe j allocate al processo i e R_{ij} il numero di risorse della classe j che il processo i può ancora richiedere al sistema:



L'invariante che deve valere ad ogni istante (per ogni $j = 1, \dots, m$) è la seguente proprietà:

$$\sum_{i=1}^n C_{ij} + A_j = E_j$$

ovvero, la somma delle risorse allocate a tutti i processi più quelle disponibili deve essere uguale al numero di risorse totali per ogni classe. A questo punto l'algoritmo funziona come segue:

1. imposta $\text{Finish}[i]$ a false per ogni $i = 1, \dots, n$

Sistemi Operativi

16 febbraio 2011

Compito

2. cerca un i tale che $R[i] \leq A$, ossia $\forall j : R_{ij} \leq A_j$
3. se esiste tale i:
 - Finish[i] viene impostato a true (ad indicare che il processo può eseguire fino alla terminazione con tutte le risorse necessarie)
 - $A = A + C[i]$ (cioè $A_j = A_j + C_{ij}$ per ogni j, ovvero, le risorse del processo selezionato vengono restituite al sistema)
 - si riprende dal punto 2
4. altrimenti, se esiste i tale che $\text{Finish}[i] = \text{false}$, allora P_i è in stallo.

L'algoritmo richiede $O(m \times n^2)$ operazioni per decidere se il sistema è in stallo.

4. Si consideri il classico problema di concorrenza dei lettori e scrittori che accedono ad una base di dati comune. Si fornisca una soluzione fair basata su monitor, assumendo che l'istruzione `signal()` risvegli *tutti* i processi in attesa.

Risposta: (5 punti)

```
monitor DataBase
    condition dbFree;
    integer nreader, waitreader, waitwriter;
    boolean readerturn, writer;

    procedure writerEnter();
    begin
        waitwriter := waitwriter +1;
        while(nreader>0 or (waitreader>0 and readerturn) or writer)
            do wait(dbFree);
        waitwriter := waitwriter-1;
        writer := true;
    end;

    procedure writerExit();
    begin
        writer := false;
        readerturn := true;
        signal(dbFree);
    end;

    procedure readerEnter();
    begin
        waitreader := waitreader +1;
        while(writer or (waitwriter>0 and not(readerturn)))
            do wait(dbFree);
        waitreader := waitreader-1;
        nreader := nreader +1;
    end;

    procedure readerExit();
    begin
        nreader := nreader-1;
        readerturn := false;
        if(nreader=0) then signal(dbFree);
    end;

    nreader := 0;
    writer := false;
    waitreader := 0;
```

Sistemi Operativi

16 febbraio 2011

Compito

```
waitwriter := 0;  
readerturn := true;  
end monitor;
```

5. (a) Si consideri la tecnica di allocazione della memoria basata su paginazione. Si descrivano le tecniche di implementazione della tabella delle pagine.
- (b) Si consideri uno spazio di indirizzi logico di 32 bit con dimensione di ciascuna pagina di 4KB e dimensione della memoria fisica di 512 MB. Quante voci sono presenti nella tabella delle pagine, nel caso di
 1. tabella delle pagine a 1 livello;
 2. tabella delle pagine a 2 livelli.

Risposta: (3+2+2 punti)

- (a) Per ridurre l'occupazione della tabella delle pagine, si può paginare la tabella delle pagine stessa. In questo modo si ottengono delle implementazioni a due o più livelli grazie alle quali si possono tenere in memoria soltanto le entry relative alle pagine effettivamente utilizzate dai processi. Ovviamente, dato che ogni livello è memorizzato in RAM, la conversione dell'indirizzo logico in indirizzo fisico può necessitare di diversi accessi alla memoria che rallentano notevolmente il sistema. Per ovviare a ciò si fa uso dei registri associativi (TLB) ed al caching degli indirizzi di pagina, riducendo così drasticamente l'impatto degli accessi multipli.
Nel caso di spazi di indirizzamento molto grandi, può essere utile avere un'implementazione della tabella delle pagine nota come tabella delle pagine invertita. In questo caso c'è una sola tabella nel sistema con un'entry per ogni frame, non per ogni pagina logica. Ogni entry consiste nel numero della pagina (virtuale) memorizzata in quel frame, con informazioni riguardo il processo che possiede la pagina. In questo modo diminuisce la memoria necessaria per memorizzare la page table, ma aumenta il tempo di accesso alla tabella. Questo schema è usato su diversi RISC a 32 bit (PowerPC), e tutti quelli a 64 bit (UltraSPARC, Alpha, HPPA, . . .), ove una page table "classica" occuperebbe diversi petabyte (es: a pagine da 4k: $8 \times 2^{52} = 32\text{PB}$ per ogni page table). Per ridurre i tempi di ricerca nella tabella invertita, si usa una funzione di hash per limitare l'accesso a poche entry (1 o 2, solitamente) nella risoluzione di un indirizzo logico.
 - (b) L'indirizzamento all'interno di una pagina di $4\text{KB} = 2^{12}$ byte richiede 12 bit. Quindi il numero di pagine logiche sarà pari a $2^{32-12} = 2^{20}$. Ciò significa che il numero di voci presenti nella tabella delle pagine sarà 2^{20} sia nel caso di una tabella delle pagine a 1 livello che nel caso di una tabella delle pagine a 2 livelli.
6. Si illustrino brevemente le differenze fra I/O ad interrogazione ciclica (Programmed I/O), I/O guidato da interrupt (Interrupt-driven I/O) e trasferimento diretto in memoria (DMA).

Risposta: (3 punti) Mentre con la modalità Programmed I/O (I/O a interrogazione ciclica) il processore manda un comando di I/O e poi attende che l'operazione sia terminata, testando lo stato del dispositivo con un loop busy-wait (polling), con la modalità Interrupt-driven I/O, una volta inviato il comando di I/O, il processo viene sospeso fintanto che non arriva un interrupt a segnalare il completamento dell'operazione. Durante la sospensione del processo, la CPU può mandare in esecuzione altri processi o thread. Di fondamentale importanza è il vettore di interrupt che consente di selezionare la routine di gestione opportuna per ogni tipo di interrupt. Ovviamente la prima modalità è efficiente soltanto nel caso in cui la velocità del dispositivo di I/O sia paragonabile a quella della CPU. La modalità DMA richiede un controller DMA e funziona in questo modo: la CPU imposta i registri del controller DMA specificando il tipo di azione di I/O, l'indirizzo di memoria ed il conteggio di byte da trasferire. Poi i dati vengono trasferiti senza più richiedere l'intervento della CPU; infatti il controller del dispositivo di I/O riceve le richieste di lettura o scrittura da parte del controller DMA a cui notifica il completamento dell'operazione una volta che ha trasferito il byte da/verso l'indirizzo di memoria corretto (specificato dal controller DMA). A questo punto il controller DMA incrementa l'indirizzo di memoria comunicandolo sul bus e decremente il conteggio dei byte da trasferire, ripetendo la richiesta di lettura o scrittura al controller del dispositivo fintanto che il conteggio dei byte non raggiungerà lo zero. Soltanto a questo punto verrà inviato un interrupt alla CPU che potrà far ripartire il processo sospeso. Siccome il controller DMA deve bloccare il bus per consentire i trasferimenti dal controller del dispositivo alla memoria, se anche la CPU ha bisogno di accedere al bus dovrà aspettare, venendo così rallentata.

Sistemi Operativi
16 febbraio 2011
Compito

7. Si spieghino brevemente le motivazioni per l'introduzione dell'architettura RAID ed il funzionamento dei livelli 0 e 1.

Risposta: (3 punti) L'architettura RAID è stata introdotta per migliorare l'affidabilità e le prestazioni del sistema dei dischi in un sistema di calcolo. Infatti, aumentando il numero di dischi fisici presenti in un calcolatore, diminuisce proporzionalmente il tempo medio fra un guasto e l'altro. Quindi il sistema RAID introduce degli schemi di ridondanza dei dati per consentire il funzionamento dei dischi anche in situazioni critiche.

Al livello 0 (striping) i dati vengono "affettati" e memorizzati sui dischi del sistema in modo parallelo. Ciò comporta un'altissima performance sia in lettura che in scrittura, ma non c'è ridondanza e quindi il guasto di un singolo disco comporta la perdita di tutti i dati.

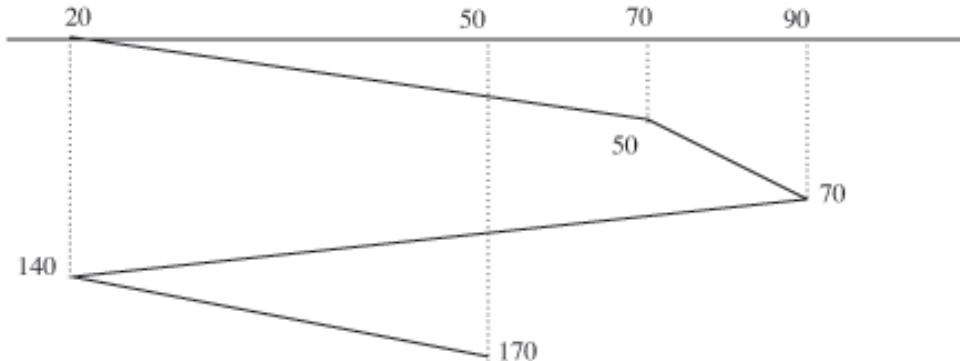
Al livello 1 (mirroring o shadowing) i contenuti di ogni disco sono interamente duplicati in un altro disco del sistema, garantendo un'eccellente resistenza ai crash, alte performance in lettura, ma non in scrittura.

8. Si consideri un disco gestito con politica C-LOOK. Inizialmente la testina è posizionata sul cilindro 20 con direzione di servizio ascendente; lo spostamento ad una traccia adiacente richiede 1 ms. Al driver di tale disco arrivano richieste per i cilindri 90, 20, 70, 50, rispettivamente agli istanti 0 ms, 10 ms, 30 ms, 80 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 quattro richieste in oggetto?

Risposta: (3+2 punti)

1. Le richieste vengono servite nell'ordine 70, 90, 20, 50:



2. Il tempo di attesa medio per le quattro richieste in oggetto è

$$\frac{(50-30)+(70-0)+(140-10)+(170-80)}{4} = \frac{20+70+130+90}{4} = \frac{310}{4} = 77,5 \text{ ms.}$$