TESTO (16-6-2020)

Descrivere lo schema di gestione della memoria basato su paginazione. Inoltre, per il caso di memoria virtuale basata su paginazione a 2 livelli, in cui ciascuna tabella delle pagine a qualsiasi livello sia costituita da 512 elementi, si determini: 1) per indirizzi logici a 40 bit la struttura dell'indirizzo e l'utilizzo dei bit nel suddetto schema di paginazione; 2) il numero massimo delle pagine fisiche (frame) di memoria impegnate da un processo nel caso in cui il numero delle tabelle a qualsiasi livello correntemente usate per quel processo in suddetto schema di paginazione sia pari a 10.

SOLUZIONE (della parte esercizio)

1. Una tabella delle pagine con 512 elementi richiede che si utilizzino 9 bit per indicizzare il singolo elemento. Essendo lo schema di paginazione a 2 livelli sono quindi necessari 9 bit per il primo livello e 9 bit per il secondo livello della tabella, quindi un indirizzo logico a 40 bit e' strutturato come 9-9-22 bit di cui i 22 meno significativi determinano l'offset di pagina

2. Dato che lo schema di paginazione a 2 livelli, e' necessario che una delle 10 tabelle sia di primo livello e quindi non permetta di giungere su frame associati alle pagine logiche, mentre le altre 9 possono essere di secondo livello, per cui il numero massimo di frame impegnati risulta essere 9 * 512

TESTO (16-6-2020)

Si consideri un insieme di 3 processi {LETT, PROD1, PROD2}. LETT accede periodicamente ad un segmento di memoria condivisa M fatto di due slot (M[1] ed M[2]) per leggerne il contenuto. PROD1 e PROD2 aggiornano periodicamente il contenuto della memora condivisa secondo il seguente schema: PROD1 aggiorna M[1] mentre PROD2 aggiorna M[2]. Lettura ed aggiornamento devono avvenire in mutua esclusione. L'aggiornamento di entrambi gli slot M[1] ed M[2] deve figurare come se fosse un'azione atomica, ovvero LETT non può leggere il contenuto della memoria condivisa M nel caso in cui sia stato aggiornato solo uno dei due slot. Inoltre, la lettura del contenuto della memoria condivisa da parte di LETT non deve essere impedita nel caso in cui solo uno tra i due processi PROD1 e PROD2 sia pronto per l'aggiornamento. Si risolva tale problema di sincronizzazione, utilizzando solo semafori, fornendo lo pseudo-codice delle procedure: AGGIORNA (usata da PROD1 e PROD2) e LEGGI (usata da LETT).

SOLUZIONE

Semaphore P1 = 0 //semaforo per indicare all'altro produttore di voler aggiornare
Semaphore P2 = 0 //semaforo per indicare all'altro produttore di voler aggiornare
Semaphore P3 = 0 //semaforo per indicare all'altro produttore di aver aggiornato
Semaphore S = 2 //semaforro per la mutua esclusione di letture e scritture

PROD1
Signal(P1)
Wait(P2)
Wait(S)
<aggiornamento di M[1]>
Signal(P3)

PROD2
Signal(P2)
Wait(P1)
Wait(S)
<aggiornamento di M[2]>
Wait(P3)
Signal(S,2)

LETT
Wait(S,2)
<lettura di M[1] e M[2]>
Signal(S,2)

TESTO (18-6-2020)

Descrivere l'algoritmo di scheduling di CPU Round-Robin (RR), discutendo l’impatto della scelta del time-slice sul comportamento dei processi I/O bound. Si consideri inoltre il caso in cui vengono attivati allo stesso istante temporale 5 processi P1, ... , P5. P1 e' un processo I/O bound che interagisce con un dispositivo D per N=10 volte e poi termina (il processo termina non appena l’ultima interazione e' completata). Tutti gli altri processi sono CPU bound di durata infinita. Supponendo che (1) la lunghezza di ogni CPU burst di P1 prima di una nuova richiesta di I/O sia nota a priori e pari a 5 ms., (2) il dispositivo D impieghi 9 ms. per ogni interazione, (3) il tempo di CPU per il dispatcher RR sia nullo, si determini (motivando la risposta) quale tra i seguenti valori del time-slice minimizza il tempo di attesa di P1 in caso di dispatching RR: 2 ms., 6 ms., 18 ms. Si calcoli inoltre il tempo di attesa almeno nel caso del time-slice che minimizza tale tempo.

SOLUZIONE (della parte esercizio)

1. Con un time slice di 2 millisec ogni richiesta di I/O da parte di P1 a partire dalla seconda ha un'attesa di 4*2+4*2+3*2+1 = 23 millisecond, c'e' poi anche l'attesa per eseguire il primo burst che e' 4*2+4*2 = 16 millisecondi (supponendo che P1 sia schedulato in CPU prima degli altri processi)

2. Con un time slice di 6 millisecondi si hanno solo attese di P1 per la gestione dei burst a partire dal secondo, e ciascuna attesa e' di 3*6+3 = 21 millisecondi

3. Con un time slice di 18 millisecondi siamo in uno scenario simile al caso 2 per cui otteniamo che per ogni burst a partire dal secondo l'attesa e' di 3*18+9 = 63 millisecondi

Il caso 2 ottimizza quindi il tempo di attesa che globalmente sara' pari a 9 * 24

TESTO (17-9-2020)

Si consideri un insieme di processi (P1 .. PN) e un ulteriore processo PROC. Si consideri inoltre una memoria condivisa M. Il processo PROC scrive periodicamente un nuovo messaggio su M, che deve essere letto da N/2 tra i processi Pi. Data una coppia di messaggi successivi m e m' scritti da PROC, m' non puo' essere letto da un processo Pi se questo processo aveva letto il messaggio m. Inoltre PROC non puo' depositare su M un nuovo messaggio se l'ultimo messaggio da lui depositato non sia stato gia' letto da N/2 tra i processi Pi. Si fornisca la soluzione di tale problema di sincronizzazione, utilizzando solo semafori, fornendo lo pseudocodice delle procedure SCRIVI e LEGGI usate, rispettivamente, da PROC e dai processi Pi.

SOLUZIONE

Semaphore S[n] //per segnalare una nuova era ai processi (P1 .. PN)
Semaphore W = 0 //per indicare la presenza di un nuovo messaggio
Semaphore L = N/2 //per indicare le letture avvenute

```
LEGGI(Pi)
Wait(S[i])
Wait(W)
<lettura del messaggio da M>
Signal(L)
```

```
Int phase = 0

SCRIVI
Wait(L,N/2)
<scrittura del messaggio su M>
Signal(W,N/2)

If phase%2 == 0
        for i from 1 to N Signal(S[i])
phase++
```

TESTO (22-1-2019)

Descrivere il funzionamento degli scheduler di CPU Shortest Process Next (SPN) e Shortest Remaining Time Next (SRTN), comparandone vantaggi e svantaggi. Inoltre, si consideri uno scenario in cui al tempo T0 nasca un processo P0 puramente CPU-bound di durata (tempo di CPU) pari a 10 secondi ed al tempo T0 + 3 secondi P0 origini un altro processo P1 puramente CPU-bound di durata (tempo di CPU) pari a 6 secondi. Supponendo che le durate dei processi siano note al tempo della loro generazione, e che il tempo di CPU per eseguire sia lo scheduler SPN che quello SRTN sia trascurabile, si calcoli il tempo massimo di completamento del processo P1 nel caso in cui il sistema abbia come scheduler SPN oppure SRTN.

SOLUZIONE (della parte esercizio)

1. Nel caso in cui lo scheduler sia SPN, questo non prevede prelazione, quindi un processo in esecuzione sulla CPU la rilascera' solo spontaneamente. In tale scenario il processo P0 non verra' prelazionato dal sistema nel momento in cui esso stesso chiedera' di attivare il processo P1. Quindi P0 terminera' la sua esecuzione esattamente all'istante di tempo T0+10 e P1 al tempo T0+10+6

2. Nel caso in cui lo scheduler sia SRTN, il sistema puo' prelazionare ogni processo correntemente in CPU non appena il software di sistema riprende il controllo. Nello scenario descritto questo avviene all'istante T0+3, quando P0 richiede (al sistema) l'arttivazione di P1. Data la regola di priorita' di SRTN, al tempo T0+3 il "remaining time" dei due processi e' 7 per P0 e 6 per P1. P0 viene effettivamente prelazionato e la CPU viene ceduta per 6 secondi a P1. Il controllo della CPU ritorna a P0 al tempo T0+3+6, e P0 terminera' quindi al tempo T0+3+6+7 = T0+16 mentre P1 al tempo T0+3+6

TESTO (22-1-2019)

Descrivere l'algoritmo ottimo di selezione della vittima per sistemi di memoria virtuale basati su paginazione. Inoltre, data una memoria di lavoro di 4 frame, e la seguente sequenza di accessi a pagine logiche, si determini quanti e quali page-fault si verificano quando è in uso l'algoritmo ottimo: 0 8 3 5 0 7 8 0 9 7 6 8 9 7 6 4

Soluzione (parte eserizio)

0 8 3 5 -> 4 fault sistemici – stato finale della memoria [0,8,3,5]
0 -> nessun fault
7 -> fault – vittime 3 oppure 5 - scegliamo 3 – stato finale della memoria [0,8,7,5]
8 0 -> nessun fault
9 -> fault – vittime 0 oppure 5 - scegliamo 0 – stato finale della memoria [9,8,7,5]
7 -> nessun fault
6 -> fault – vittima 5 – stato finale della memoria [9,8,7,6]
8 9 7 6 -> nessun fault
4 -> fault – vittima qualsiasi FAULT TOTALI 4+4

TESTO (18-2-2021)

Si consideri un insieme di processi {P1,...,Pn} che periodicamente scrivono un nuovo messaggio su una memoria condivisa M. Un ulteriore processo LETT legge un nuovo messaggio da M periodicamente rimanendo bloccato in attesa del messaggio qualora non fosse stato scritto. Ogni processo Pi che tenta di scrivere un nuovo messaggio rimane anche esso bloccato in attesa che l'ultimo messaggio scritto su M sia stato letto. Inoltre, un ulteriore processo RAND, assegna la possibilita' di scrivere un nuovo messaggio ad un processo Pi scelto a caso, solo dopo che l'ultimo processo precedentemente scelto abbia depositato il proprio messaggio. Risolvere tale problema di sincronizzazione, utilizzando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI, LEGGI e SELEZIONA utilizzate, rispettivamente, dal generico processo Pi da LETT e da RAND.

SOLUZIONE

Semaphore S[n] = [0 .. 0]//per segnalare la possibilita' di scrivere ai processi (P1 .. Pn)
Semaphore W = 0 //per indicare la scrittura avvenuta di un nuovo messaggio
Semaphore L = 1 //indica la lettura
Semaphore W1 = 1 //indica che non ci sono scritture ancora non effettuate

LEGGI
Wait(W)
<lettura del messaggio da M>
Signal(L)

SELEZIONA
X = random in 1-n
Wait(W1)
Signal(S[i])

SCRIVI (Pi)
Wait(S[i])
Wait(L)
<scrittura del messaggio su M>
Signal(W)
Signal(W1)

TESTO (22-1-2019)

Si consideri un sistema di N processi concorrenti {P1, ..., PN}, i quali accedono periodicamente in lettura ad un segmento di memoria condivisa M. Si consideri inoltre un processo PROD che accede al segmento M in scrittura. Periodicamente PROD scrive su M un nuovo messaggio, e questo può accadere solo dopo che l'ultimo messaggio da lui scritto sia stato letto K volte (con $K \leq N$) da generici processi nell'insieme {P1, ..., PN}. D'altro canto, ogni processo Pi deve rimanere temporaneamente bloccato nel caso in cui PROD non abbia scritto alcun messaggio su M, oppure l'ultimo messaggio scritto da PROD sia stato già letto K volte da processi nell'insieme {P1, ..., PN}. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI, usata dal processo PROD, e LEGGI usata da ognuno dei processi nell'insieme {P1, ..., PN}.

SOLUZIONE

Semaphore W = K;//semaforo che codifica la condizione di nuova scrittura ammissibile
Semaphore R = 0;//semaforo che codifica la condizione di nuove letture ammissibili

SCRIVI:
wait (W,K);
<post message to M>;
signal(R,K);

LEGGI:
wait (R,1);
<get message from M>;
signal(W,1);

TESTO (18-2-2019)

Descrivere il metodo sequenziale indicizzato per l'accesso ai file. Si consideri inoltre un file-system in cui i file sono allocati secondo uno schema contiguo, ed il dispositivo di memoria di massa abbia blocchi di capacità pari a 4096 record. Si determini la latenza massima di accesso ad un record di un file F sequenziale indicizzato che abbia taglia pari a 1M record, ed il cui file di indici f abbia taglia pari a 512 record, e contenga 128 chiavi. Per semplicità si supponga che il costo di gestione di ogni chiave, una volta caricata in memoria di lavoro, sia costante e pari ad 1 millisecondo e che il costo di caricamento di un blocco di dispositivo in memoria di lavoro sia anche esso costante e pari a 10 millisecondi.

SOLUZIONE

Il caricamento del file degli indici in memoria e la scansione delle chiavi presenti nell'indice comporta una latenza pari a L = 10 + 128x1 millisecondi

Il file F necessita per la sua allocazione di 2^20 / 4x2^10 = 2^8 = 256 blocchi

IPOTESI 1: le chiavi dell'indice sono equidistribuite, in tal caso l'ultima chiave del file indice f punta ad una zona del file fatta di 256/128 = 2 blocchi per un totale di 2x4096 record, essendo il file allocato in modo contiguo il tempo per il caricamento dei blocchi sara' semplicemente 2x10 millisecondi, e quello di accesso all'ultimo record della zona caricata sara' 2x4096 millisecondi, per un totale di 2x(5006) + L millisecondi

IPOTESI 2: distribuzione delle chiavi di caso peggiore (ovvero sui primi 128 record del file), in tal caso i blocchi andranno caricati tutti, ed andranno lette tutte le chiavi del file a partire dalla 129 per una latenza totale di 256x10 + (2^20 – 128)x1 + L

TESTO (18-2-2019)

Si consideri un sistema con un processo A ed un insieme di 4 processi {B1, B2, B3, B4}. Il processo A scrive periodicamente un nuovo messaggio su una memoria condivisa M. Dato un messaggio scritto da A, questo dovrà essere letto da esattamente 2 tra i processi Bi (senza un particolare ordine), mentre il successivo messaggio dovrà essere letto dagli altri due processi Bi. Ogni processo Bi che tenti di leggere un messaggio deve entrare in blocco nel caso in cui il messaggio non sia disponibile, oppure nel caso in cui il messaggio correntemente scritto non possa essere letto da Bi poichè lo stesso Bi aveva letto il precedente messaggio scritto da A. D'altro canto, il processo A non potrà scrivere un nuovo messaggio sulla memoria condivisa M, dovendo così entrare in blocco, prima che l'ultimo messaggio da esso scritto non sia stato letto da almeno 2 processi Bi. Si schematizzi la soluzione a tale problema di sincronizzazione usando solo semafori, fornendo il pseudo-codice della della procedura SCRIVI utilizzata dal processo A e quello della procedura LEGGI usata dal generico processo Bi.

SOLUZIONE

Semaphore S[4] = {0,0,0,0};//semafori binari per ammettere letture su fasi fatte da 2 scritture
Semaphore R = 0;//semaforo per ammettere le 2 letture in ogni fase di scrittura
Semaphore W = 2;//semaforo per ammettere una scrittura successiva

LEGGI (Bi):
wait (S[i]);
wait (R);
<read message from M>;
signal (W);

SCRIVI:
static int phase = 0;
wait(W,2);
<post message to M>;
If (!phase) for i = 1 to 4 signal (S[i]);
signal (R,2);
phase = (phase+1) mod 2;

TESTO (17-7-2019)

Si consideri un sistema di 3 processi concorrenti, A, B e C, i quali accedono ad un segmento di memoria condivisa M. Periodicamente i processi A e B aggiornano, rispettivamente, la prima metà e la seconda metà del segmento di memoria condivisa. Quando l'aggiornamento è stato eseguito, il processo A riscrive il contenuto dell'intero segmento M invertendo l'ordine dei byte. Il processo C legge periodicamente l'intero contenuto del segmento di memoria condivisa M; la lettura è abilitata solo dopo che l'aggiornamento da parte di A e B, e l'inversione da parte di A, siano stati effettuati. Nessun nuovo aggiornamento può avvenire su M da parte di A e B prima che C abbia letto il contenuto registrato in M. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI, usata dal processo B, SCRIVI –INVERTI, usata dal processo A, e LEGGI, usata dal processo C.

SOLUZIONE

Semaphore SA = 1;//semaforo per abilitare la scrittura del processo A
Semaphore SB = 1; //semaforo per abilitare la scrittura del processo B
Semaphore SC = 0; //semaforo per abilitare la lettura del processo C
Semaphore INV = 0; //semaforo per abilitare l'inversione dei byte da parte del processo B

SCRIVI:
wait (SB);
<post data to bottom half of M>
signal (INV);

SCRIVI-IMVERTI:
wait (SA);
<post data to top half of M>
wait (INV);
<revert byte order in M>;
signal (SC);

LEGGI:
wait(SC);
<read data from M>;
signal(SA);
signal(SB);

TESTO

Descrivere il metodo di allocazione di file basato su indicizzazione a livelli multipli. Si consideri un file di dati **F** di 2048 record ed un dispositivo di memorizzazione di massa con blocchi di 256 record avente tempo di accesso ai blocchi fisso pari a 30 microsecondi. Supponendo che (i) gli indici abbiano taglia pari ad un record, (ii) gli indici di primo livello siano 6, e quelli di secondo livello siano 2, (iii) il tempo di identificazione di un riferimento ad un blocco di dispositivo sia costante e pari a 5 microsecondi, calcolare il tempo per la lettura di tutto il file **F**.

Soluzione (della parte esercizio)

- numero di blocchi dati allocati per il file = 2048/256 = 8

- 6 degli 8 blocchi dati sono puntati in modo diretto dagli indici di primo livello mantenuti nell'RS. Il loro tempo di lettura è pari a 6 x (5 + 30) microsec.

- i rimanenti due blocchi dati devono essere individuati tramite indici accessibili al secondo livello, tali indici sono memorizzati in un unico blocco il cui tempo di lettura risulta (5 + 30) microsec.

- letto il precedente blocco, il tempo di lettura dei due blocchi dati puntati dagli indici accessibili al secondo livello risulta pari a 2 x (5 + 30) microsec.

tempo totale per la lettura di F = 6 x (5 + 30) + (5 + 30) + 2 x (5 + 30) microsec.

TESTO

Si descriva lo scheduler di CPU Windows. Si consideri inoltre uno scenario single-core in cui esistono solo 3 processi attivi A, B e C, tutti single-thread/CPU bound di durata 100 millisecondi, che eseguono solo in user mode, di cui i primi due di classe real-time (stesso livello di priorita' dei thread) ed il terzo di classe variable. Supponendo che il tempo T per un cambio di contesto tra i due processi (qualora realmente eseguito) sia pari a 10 microsecondi, e considerando che il software di gestione dell'interrupt da timer per lo scheduling della CPU e' tutto eseguito in kernel mode (e ad ogni interrupt e' ammesso che possa avvenire un context switch), si determini il minimo periodo dell'interrupt da timer per lo scheduling di CPU che garantisca che almeno il 90 per cento del tempo di CPU speso per i tre processi sia impegnato in user mode.

Soluzione

Parte A: descrizione dei livelli di priorita' dello scheduler windows, e della logica di assegnazione (dinamica e/o basata su system call) delle priorita' ai processi/thread attivi.

Parte B: i 2 processi a priorita' real-time vanno in time-sharing sulla CPU, il terzo viene schedulato solo a valle del loro completamento, e non subisce context switch. Bisogna quindi calcolare il numero massimo di context switch N che occorrono durante l'esecuzione dei primi 2 in modo da soddisfare la seguente equazione

$$(Ttot - Tuser)/Ttot <= 0.1$$

Ma Tuser = 300 millisec

Ttot = Tuser + N x Tsched

dove Tsched = 0.01 millisec quindi N <= 3333.3...

In conclusione Tinterrupt >= (200 / N) + 0.01 millisec ~ 0.07 millisec

TESTO

Descrivere il metodo di accesso sequenziale indicizzato ad un file. Si consideri un file di dati **F** di 2048 record ed il relativo file di indici **f** contenente 1024 elementi. Supponendo che il tempo di accesso ad un indice del file **f** sia 5 microsecondi e che il tempo di accesso di ogni singolo record del file **F** sia 20 microsecondi, calcolare il tempo massimo per l'accesso ad un record del file **F** con chiave generica.

Soluzione (della parte esercizio)

- il tempo massimo si ottiene quando:
  1. la chiave ricercata è associata all'ultimo record del file dati **F**
  2. si ha una configurazione di caso peggiore del file di indici **f**

- la configurazione peggiore si ha quando la chiave ricercata tra le 2048 è nell'intervallo associato all'ultimo elemento del file **f** e tale intervallo ha la massima ampiezza, ovvero quando esistono 1023 intervalli di ampiezza 1 ed un unico intervallo di ampiezza 1025

- per giungere all'ultimo record di suddetto intervallo devo scandire tutto il file degli indici, questo porta via un tempo pari a 5 x 1024 devo poi leggere tutti i blocchi dell'intervallo, questo porta via un tempo 20 x 1025

tempo massimo per l'accesso ad un record di F con chiave generica = 5 x 1024 + 20 x 1025 microsecondi

TESTO

Si descriva l'allocazione dei file a catena. Si consideri inoltre un file system che usa l'allocazione a catena. Dato un file di taglia pari a 10 MB, puntatori a blocchi di disco di taglia pari a 32 byte, e organizzazione dei record variabile con riporto, si determini il numero di blocchi di disco necessari a memorizzare il file nei seguenti casi: (i) dimensione del blocco di disco 512 byte; (ii) dimensione del blocco di disco 1024 byte.

Soluzione (della parte esercizio)

- la taglia globale del file e' pari a
  10 x 2^20 = N bytes

- di ogni blocco di disco 32 byte non sono disponibili per la memorizzazione dei dati del file otteniamo quindi i seguenti 2 casi

- caso di il blocchi di disco pari a 512 byte:
  - Il numero di blocchi necessari a memorizzare il file e' pari a
    N/(512 – 32) parte intera superiore = 21846

- caso di il blocchi di disco pari a 1024 byte:
  - Il numero di blocchi necessari a memorizzare il file e' pari a
    N/(1024 – 32) parte intera superiore = 10571

TESTO

Descrivere l'algoritmo di scheduling della CPU "multi-level feedback queue". Si consideri inoltre il caso in cui venga attivato all'istante T un processo P CPU bound, che richiede 30 unità di tempo di CPU per completare la sua esecuzione. Supponendo che (1) all'istante di tempo $T' > T$ vengano attivati $N \geq 1$ processi I/O bound (2) il quanto di tempo Δ per il livello di priorità 0 (livello di priorità massimo usato come livello di ammissione) sia pari ad 1 unità di tempo di CPU (3) l'attivazione e lo scheduling dei processi abbia costo nullo (4) la preemption su un processo in esecuzione avvenga solo allo scadere del relativo quanto di tempo, **determinare:** il minimo valore di T' affinché il processo P completi la sua esecuzione entro il tempo T+30 nel caso in cui i livelli di priorità dello scheduler di CPU siano 3.

Soluzione

- per poter completare entro l'istante T+30, ed avendo durata pari a 30 unita' di tempo di CPU, il processo P non deve mai essere deschedulato

- quindi il suo ultimo quanto di tempo dovra' iniziare prima del tempo T'

- avendo lo scheduler 3 livello di priorita', P spendera' 1 quanto di 1 unita' a livello 0, 1 quanto di 2 unita' a livello 1, e 27/4 = 6.75 quanti al livello 2

- l'ultimo di questi quanti (a livello 2) verra' attivato all'istante 1+2+(6x4) = 27

- quindi il valore di T' che soddisfa lo scheduling e T' > 27

TESTO

Descrivere l'algoritmo di schedulazione del disco SSTF, discutendone vantaggi e svantaggi. Si consideri inoltre la sequenza di operazioni di I/O corrispondenti alle tracce: 16, 43, 9 e 91. Le operazioni di I/O vengono richieste nell'ordine delle tracce nella sequenza. Supponendo che (1) il disco abbia 100 tracce (numerate da 1 a 100), (2) la testina sia inizialmente posizionata sulla traccia 67, (3) la sequenza di scheduling comprenda inizialmente solo le richieste per le tracce 16, 43 e 9, deteminare l'istante di inserimento ottimale della richiesta per la traccia 91 (rispetto al servizio per le richieste già presenti) in modo da minimizzare lo spostamento della testina.

Soluzione (della parte esercizio)

- i casi possibili sono 4

La richiesta per la traccia 91 e' gia' presente nella sequenza iniziale di scheduling, in tal caso otteniamo che le richieste vengono servite in questo ordine: 43 16 9 91 (movimento totale = 140) OPPURE 91 43 16 9 (movimento totale = 106)

La richiesta per la traccia 91 non e' presente prima i servire la prima delle richieste gia' pesenti, in tal caso otteniamo che le richieste vengono servite in questo orine: 43 16 9 91

La richiesta per la traccia 91 non e' presente prima di servire le prime due richieste gia' presenti, in tal caso otteniamo che le richieste vengono servite in questo orine: 43 16 9 91

La richiesta per la traccia 91 non e' presente prima di servire tutte le richieste gia' presenti, in tal caso otteniamo che le richieste vengono servite in questo orine: 43 16 9 91

Di fatto la sequenza di scheduling non cambia mai (eccetto che nello scenario 1 se preferiamo la traccia 91 alla 43 nello scheduling), ed il movimento della testina e' sempre lo stesso in tutti i casi

TESTO

Si descrivano i meccanismi di paginazione a livelli multipli ed i vantaggi che essi offrono nei sistemi di memoria virtuale. Si consideri inoltre un sistema di paginazione a 2 livelli con indirizzi logici di 32 bit, dei quali 5 vengono usati per identificare la sezione e 5 per identificare la pagina. Si consideri inoltre un processo che abbia due zone di indirizzi logici contigui mappate e materializzate: la zona tra gli indirizzi [0,1GB) e la zona tra gli indirizzi [2GB,3GB). Si identifichi il numero di sezioni valide nella tabella delle pagine.

Soluzione (della parte esercizio)

- Con 32 bit di indirizzamento abbiamo una memoria logica di 4GB
- Con 5 bit per il numero di sezione abbiamo 2^5 = 32 sezioni
- Ciascuna sezione copre 4/32GB = 1/8 GB
- Per coprire il mapping in memoria fisica di 1GB abbiamo quindi bisogno di 8 sezioni
- Le sezioni compaiono in ordine nello spazio di indirizzamento per cui
  - ✓ l'intervallo i indirizzi [0-1GB) e' coperto dalle sezioni 0-7
  - ✓ l'intervallo i indirizzi [2GB-3GB) e' coperto dalle sezioni 16-23

In conclusione, per avere gli ntervalli di indirizzi logici [0-1GB) e [2GB-3GB) materializzati la tabella delle pagine di primo livello deve avere le validi gli intervali di elementi 0-7 e 16-23

TESTO

Si considerino N processi scrittori (S1 ... Sn) ed M processi lettori (L1 ... Lm) i quali accedono ad uno stesso segmento di memoria condivisa MEM, il quale ha la capienza di un messaggio. Ogni processo scrittore Si deposita periodicamente un nuovo messaggio su MEM, ed attende la risposta (sempre scritta su MEM) da uno qualsiasi dei processi lettori Li. Questi ultimi verificano periodicamente la presenza di nuovi messaggi su MEM, ed entrano in attesa in caso nessun nuovo messaggio sia presente. D'altra parte, ogni processo scrittore che intenda depositare un nuovo messaggio entra in attesa in caso un messaggio precedentemente depositato da un qualsiasi processo scrittore non abbia ancora avuto risposta. Si schematizzi la soluzione di tale problema di sincronizzazione, utilizzando solo semafori, costituita dalla traccia (in termini di pseudo-codice) delle procedure: **DEPOSITA-MESSAGGIO** (usata da Si) e **LEGGI-RISPONDI** (usata da Li).

Soluzione

- Il buffer MEM deve essere acceduto in modo esclusivo da Si e poi da uno dei processi Li per eseguire lo scambio di informazioni
- Serve quindi un mutex (o semafioro binario) M1 per l'accesso da parte di Si (fino alla chiusura della cooperazione con Li) e poi un mutex (o semaforo binario) M2 per indicare a Li che il messaggio e' stato scritto
- Serve anche un mutex (o semaforo binario) M3 per indicare a Si che la risposta e' stata scritta da Li (questo non puo' concidere con M1 altrimenti violerei la mutua esclusione nell'accesso a MEM in scrittra/lettura)

Quindi:

DATA: mutex M1=1, M2 = 0, M3 = 0;

DEPOSITA-MESSAGGIO

wait(M1)
<write message into MEM>
signal(M2)
wait(M3)
<read reply from MEM>
signal(M1)

LEGGI-RISPONDI

wait(M2)
<read message from MEM>
<write reply into MEM>
signal(M3)

TESTO

Si consideri un insieme di N processi (P0, ... , PN-1), ciascuno dei quali scrive periodicamente un nuovo messaggio su una memoria condivisa M di N slot. Ciascun processo Pi scrive esclusivamente nell'i-esimo slot della memoria condivisa. Dopo aver scritto il proprio messaggio, Pi attende che venga consegnata una risposta tramite un buffer R, a slot singolo. L'attesa della risposta e' bloccante. Un ulteriore processo REPLY legge i messaggi prodotti dai processi Pi in ordine, ovvero a partire da P0 fino a PN-1, e poi ancora da P0. La lettura e' anche in questo caso bloccante. Quando REPLY ha letto un nuovo messaggio proveniente da un processo Pi, consegna ad esso la risposta tramite il buffer R, la quale potra' essere letta solo da quel processo. Gli accessi in scrittura su M da parte dei processi (P0, … , PN-1) potranno avvenire anche essi in concorrenza. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure **SCRIVI** e **RISPONDI** usate rispettivamente da Pi e REPLY

Soluzione

- E' necessario un array di mutex (o semafori binari) MESS[] per indicare a REPLY che ogni processo Pi ha scritto un nuovo messaggio sul suo slot in M (i.e. M[i])
- E' necessario un array di mutex (o semafori binari) REP[] per indicare a ciascun processo Pi che la sua risposta in R e' pronta
- E' necessario un mutex (o semaforo binario) FRE-R per indicare a REPLY che il buffer R puo' essere sovrascritto con una nuova risposta

Quindi:

DATA: mutex MESS[N] = {0...0} , REP[N] = {0...0}, FRE-R = 1;

SCRIVI

```
Static int i = 0;
wait(MESS[i]);
<read message from slot M[i]>;
wait(FREE-R);
<write reply into R>;
signal(REP[i]);
i  = (i+1)mod(N);
```

LEGGI-RISPONDI (Pi)

```
<write message into M[i]>;
signal(MESS[i]);
wait(REP[i]);
<read reply from R>;
signa(FREE-R);
```

TESTO

Considerare un sistema formato da tre processi concorrenti che competono nell'accesso ai MUTEX M1, M2 ed M3, ed al semaforo S (inizializzato al valore 2) per gestire delle sezioni critiche. Determinare se è possibile o no il verificarsi di un deadlock, motivando la risposta. In caso affermativo, fornire una riscrittura della sequenza degli statement dei processi che garantisca lo stesso livello di isolamento delle regioni (o sezioni) critiche, ma che sia esente da deadlock.

Mutex M1, M2, M3; Semaphore S⇐2;

```
P1:                    P2:                    P3:
wait(S,2);             wait(S);               wait(S);
CriticalReg();         CriticalReg();         CriticalReg();
Signal(S,2);           MutexLock(&M2);        MutexLock(&M3);
MutexLock(&M1);        MutexLock(&M3);        MutexLock(&M1);
MutexLock(&M2);        NestedCriticalRegion(); NestedCriticalRegion();
CriticalRegion();      MutexUnlock(&M3);      MutexUnlock(&M1);
MutexUnlock(&M2);      MutexUnlock(&M2);      MutexUnlock(&M3);
MutexUnlock(&M1);      signal(S);             signal(S);
```

Soluzione

Mutex M1 = 1, M2 = 1, M3 = 1; Semaphore S= 2;