SCRITTO D’ESAME DEL 13/6/2022

Si consideri un insieme di N processi (P1 , ..., PN) ed un altro insieme di M processi (L1 , ..., LM). Ogni processo Pi scrive periodicamente un nuovo messaggio in una memoria condivisa M, che deve essere letto una sola volta da tutti i processi Lj.

Il processo Pi può scrivere un nuovo messaggio solo dopo che l’ultimo messaggio scritto sia stato letto da tutti i processi Li. Altrimenti dovrà rimanere in attesa.

Allo stesso tempo, un processo Pi che ha scritto un messaggo in M ne può scrivere un successivo solo dopo che anche tutti gli altri processi Pj (con j diverso da i) abbiano scritto il loro messaggio in M. Altrimenti dovrà rimanere in attesa.

Allo stesso tempo, ogni processo Lj che intende leggere dovrà rimanere in attesa che un messaggio non ancora letto da Lj sia reso disponibile.

Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI e LEGGI usate rispettivamente dai generici processi Pi e Lj.

NON COPERTO IL BOLD DELLA DOMANDA

Semaphore S-WRITE = 1; //serializziamo le scritture
Semaphore DATA-READ = M; //serializziamo scrittura e letture
Semaphore CAN-READ[M] = { 0, ..., 0}; //attendiamo la consegna del messaggio per la lettura – indicizzato a partire da 1

SCRIVI (Pi)
wait(S-WRITE);
for j = 1 to M wait(DATA-READ);
<write data on M>;
for j = 1 to M signal(CAN-READ[j]);
signal(S-WRITE)

LEGGI (Lj)
wait(CAN-READ[j]);//j is the process index
<read data from M>;
signal(DATA-READ)

COPERTO IL BOLD DELLA DOMANDA

Semaphore S-WRITE = 1;
Semaphore DATA-READ = M;
Semaphore CAN-READ[M] = { 0, .., 0}; //indexed starting from 1
Semaphore ALL-DONE[N] = { N, …, N}; //indexed starting from 1

SCRIVI (Pi)
for j = 1 to N wait(ALL-DONE[i]);//take all tokens for the same semaphore with index i (for Pi)
wait(S-WRITE);
for j = 1 to M wait(DATA-READ);
<write data on M>;
for i = j to M signal(CAN-READ[j]);
signal(S-WRITE)
for j = 1 to N signal(ALL-DONE[j]);//put one token on each semaphore

LEGGI (Lj)
wait(CAN-READ[j]);
<read data from M>;
signal(DATA-READ);

SCRITTO D’ESAME DEL 26/1/2023

Si considerino un insieme di N processi {P1, P2, ... , PN} e due ulteriori processi PROD1 e PROD2. Ogni processo Pi periodicamente vorrá acquisire un nuovo messaggio di PROD1 oppure di PROD2 leggendolo da una memoria condivisa R, e rimarrá in attesa fino a che non sara’ riuscito a leggere tale nuovo messaggio.

I processi PROD1 e PROD2 periodicamente attendono che tutti gli N processi Pi generici siano pronti a leggere un nuovo messaggio. Questo abilita la consegna del nuovo messaggio sulla memoria condivisa R solo da parte di uno dei due processi PROD1 e PROD2.

In ogni caso, per ogni coppia di nuovi messaggi consegnati ai processi Pi, la consegna di uno (non necessariamente il primo) dei messaggi deve essere effettuata da PROD1 e dell’altro dei messaggi da PROD2.

Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure LEGGI usata dai processi Pi e SCRIVI usata dai processi PROD1 e PROD2.

NON COPERTO IL BOLD DELLA DOMANDA

Semaphore Lettura_abilitata[N] = {0, ..., 0};//indexed starting from 1
Semaphore Write = 1;
Semaphore Waiting_readers = 0;

LEGGI (i)
Signal(Waiting_readers);
Wait (Lettura_abilita[i]);
< read from M>;

SCRIVI (j)
Wait(Write);
For i = 1 to N Wait (Waiting_readers);
< write to M>;
For i = 1 to N Signal(Lettura _abilitata[i]);
Signal(Write);

COPERTO IL BOLD DELLA DOMANDA

Semaphore Lettura_abilitata[N] = { 0, ..., 0};//indexed starting from 1
Semaphore Write = 1;
Semaphore Waiting_readers = 0;
Semaphore Can_write[2] = {2, 2};//indexed starting from 1

LEGGI (i)
Signal(Waiting_readers);
Wait (Lettura_abilita[i]);//semaphore indexed starting from 1
< read from M>;

SCRIVI (j)
for x = 1 to 2 Wait(Can_write[j]);
Wait(Write);
For i = 1 to N Wait (Waiting_readers);
< write to M>;
For i = 1 to N Signal(Lettura _abilitata[i]);
Signal(Write);
For x = 1 to 2 signal(Can_write[x]);

SCRITTO D’ESAME DEL 8/9/2022

Si descriva lo scheduler di CPU multi-level feedback-queue. Inoltre, si consideri uno scenario in cui all’istamte T0 esistano 2 processi P1 e P2.

Il processo P1 é CPU-bound di durata infinita, mentre il processo P2 é I/O-bound ed interagisce infinite volte con un dispositivo D prima di completare la sua esecuzione.

Il CPU-burst di P2 sia di 5 millisecondi. Nel caso lo scheduler multi-level feedback-queue abbia 2 livelli di prioritá, si determini motivando la risposta quale sia il massimo valore di time-slice che garantisca un tempo di attesa di P2 non superiore a 10 millisecondi.

Si assuma che la latenza di esecuzione dello scheduler di CPU sia nulla.

Considerando una strutturazione classica dello scheduler multilevel-feedback queue
si puo' supporre che

- TS sia il time-slice al livello di priorita' 0
- 2 x TS sia il time-slice al livello di priorita' 1

Il tempo di risposta del dispositivo D e' di fatto arbitrario (poiche' non ci sono indicazioni o ipotesi relative ad esso all'interno del testo)

Esistendo solo 2 processi (P1 e P2) se questi sono entrambi non-blocked l'assegnazione della CPU avviene in modo alternato ad ogni specifico livello di priorita'

Il tempo di attesa massimo e' quindi determinabile in base al massimo utilizzo del time-slice da parte dei processi in quel livello di priorita', pertanto se P2 ritorna nello stato ready in quel livello di priorita' quello che determina il tempo di attesa e' il residuo di utilizzo del time-slice da parte di P1

Il rientro nello stato ready da parte di P2 avviene in un istante arbitrario, quindi nel caso peggiore otteniamo di dover attendere per tutto il time-slice, che nel caso di priorita' 1 e' pari a (2 x TS) ne deriva quindi che $(2 \times TS) < 10$ quindi $TS < 5$

Tale disequazione determina anche che transiti **wait→ ready** da parte di P2 avverranno solo quando P2 si trovera' a livello di priorita' 1, che e' esattamente il livello massimo, per il quale si ha la predetta alternanza di esecuzione di processi non-blocked

SCRITTO D’ESAME DEL 21/6/2021

Si consideri un insieme di N processi {P0, P1, P2, P3, ..., PN-1}, e una memoria condivisa M composta da N/2 slot.

Il processo P[i] periodicamente attende che un nuovo messaggio venga depositato su M[(i mod(N/2))].

Un ulteriore processo PROD scrive periodicamente messaggi su M secondo uno schema buffer circolare. Ogni messaggio scritto da PROD in M[i] può essere letto da un solo processo dell’insieme {P0, P1, P2, P3, ..., PN-1}. **Inoltre, ogni processo Pi non può leggere entrambi i messaggi di una coppia di messaggi consecutivi depositati da PROD in uno stesso slot di M.**

Se all’atto della lettura Pi non ha la possibilità di leggere il messaggio correntemente nello slot oppure un messaggio da lui leggibile non sia presente, Pi deve entrare in stato di attesa. Inoltre, PROD deve entrare in stato di attesa se all’atto della scrittura il corrispettivo slot di M contenga un messaggio non ancora letto.

Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI e LEGGI usate, rispettivamente, da PROD e da ciascuno dei processi Pi.

NON COPERTO IL BOLD DELLA DOMANDA

Semaphore Scrittura_abilitata[N/2] = { 1, …, 1};
Semaphore Lettura_abilitata[N/2] = { 0, …, 0};

integer i = 0;
SCRIVI
Wait(Scrittura_abilitata[i]);
<write to M[i]>;
Signal(Lettura_abilitata[i]);
i = (i + 1) mod N/2

LEGGI (i)
Wait(Lettura_abilitata[i mod N/2]);
<read from M[i]>;
Signal(Scrittura_abilitata[i mod N/2]);

COPERTO IL BOLD DELLA DOMANDA

Semaphore Scrittura_abilitata[N/2] = { 1, …, 1};
Semaphore Lettura_abilitata[N/2] = { 0, …, 0};
Semaphore Can_read[N] = {2, …, 2};

integer i = 0;
SCRIVI
Wait(Scrittura_abilitata[i]);
<write to M[i]>;
Signal(Lettura_abilitata[i]);
i = (i + 1) mod N/2

LEGGI (i)
for x = 1 to 2 Wait(Can_read[i]);
Wait(Lettura_abilitata[i]);
<read from M[i]>;
Signal(Scrittura_abilitata[i]);
Signal(Can_read[i]);
Signal(Can_read[(i + N/2) mod N]);

SCRITTO D’ESAME DEL 16/9/2021

Si descriva il metodo di allocazione dei file indicizzato. Si consideri un file system con metodo di allocazione dei file indicizzato a 2 livelli. Il record di sistema che tiene traccia di ogni file ha 6 indici diretti e 4 indici indiretti.

Il dispositivo di memoria di massa che ospita il file system ha blocchi di taglia pari a 2048 byte. Si calcoli la taglia massima, espressa in byte, di un file gestibile su tale file system considerando i due scenari dove

1) l’indice che identifica un blocco all’interno del dispositivo abbia taglia pari a 8 byte, oppure

2) abbia taglia pari a 4 byte.

La porzione del file gestita tramite indici diretti e' indipendente dalla taglia dell'indice (dipende escusivamente dalla taglia del blocco del dispositivo), in particolare tale porzione P del file ha taglia massima pari a

P = 6 x 2048 byte

Per la porzione del file gestita tramite indici indiretti la taglia dipende dalla taglia degli indici di secondo livello poiche' il loro numero e' funzione della taglia del blocco

Abbiamo quindi due scenari

1) indici di taglia pari a 8 byte – in questo caso ciascuno dei 4 blocchi del dispositivo che mantengono indici di secondo livello ne ha 2048/8 = 256 permettendo una taglia della porzione P' di file gestibile pari a P' = 4 x 256 x 2048 byte, per un totale della taglia del file di P + P' = 6 x 2048 + 4 x 256 x 2048 byte

2) indici di taglia pari a 4 byte – in questo caso ciascuno dei 4 blocchi del dispositivo che mantengono indici di secondo livello ne ha 2048/4 = 512 permettendo una taglia della porzione di file gestibile di P' = 4 x 512 x 2048 byte, per un totale della taglia del file di P + P' = 6 x 2048 + 4 x 512 x 2048 byte

SCRITTO D’ESAME DEL 16/9/2021

Si descriva lo scheduler di CPU round-robin virtuale. Inoltre, supponendo che il quanto di tempo di tale scheduler sia pari a 3 millisecondi e che al tempo T = 0 siano creati in ordine 2 processi P1 e P2 tali che P1 abbia un solo CPU-burst di durata pari a 10 millisecondi, mentre P2 abbia due differenti CPU-burst di durata pari a 2 e 4 millisecondi e al termine del primo burst rimanga in stato bloccato per 1 millisecondo mentre al termine del secondo burst termini, si determini il tempo di completatmento di P2.

Creazione in ordine presuppone che P1 sia nella ready queue di livello NORMAL come primo processo

P1 usera' quindi 3 millisecondi di CPU e si riaccodera' come ready nella stessa coda

Al tempo 3 P2 usera' 2 millisecondi di CPU e andra' in blocco **(fino al tempo 5 + 1), e al risveglio verra' inserito nella coda a priorita' HIGH, avendo un residuo di 1 millisecondo**

Al tempo 5 (ovvero 3 + 2) il processo **P1 usera' 3 millisecondi** e si riaccodera' come ready nella coda di livello NORMAL

Al tempo al tempo 8 (ovvero 5 + 3) il processo P2 prendera' la CPU per usarla per il residuo non usato in precedenza, ovvero 1 milisecondo

Al tempo 9 (ovvero 8 + 1) il processo **P1 prendera' la CPU per 3** millisecondi e poi si riaccodera' come ready nella queue a priorita' NORMAL

Al tempo 12 (ovvero 9 + 3) il processo P2 prendera' la CPU per eseguire tutto il time-slice di 3 millisecondi realtivo alla priorita' NORMAL, **teminando al tempo 12+3 = 15 millisecondi**

PROVA D’ESAME DEL 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).

NON COPERTO IL BOLD DELLA DOMANDA

Semaphore Lettura_abilitata = 2;
Semaphore Update_done[2] = {0, 0}

LEGGI
Wait (Lettura_abiltata, 2);
<read from M>;
Signal (Lettura_abilitata, 2);

AGGIORNA (PROD i/j – ½)
Wait (Lettura_abilitata);
<aggiorna M[i]>;
Signal(Udate_done[j]);
Wait(Update_done[i]);
Signal(Lettura_abilitata);

COPERTO IL BOLD DELLA DOMANDA

Semaphore Lettura_abilitata = 2;
Semaphore Update_done[2] = {0, 0};
Semaphore Ready_to_update[2] = {0, 0};//indexed starting from 1

LEGGI
Wait (Lettura_abiltata, 2); //azione atomica per il prelievo di 2 token
<read from M>;
Signal (Lettura_abilitata, 2);//azione atomica per il rilascio di 2 token

AGGIORNA (PROD i/j – ½)
Signal(Ready_to_update[j]);
Wait(Ready_to_update[i]);
Wait (Lettura_abilitata);
<aggiorna M[i]>;
Signal(Udate_done[j]);
Wait(Update_done[i]);
Signal(Lettura_abilitata);

PROVA D’ESAME DEL 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.

Paginazione a 2 livelli implica che esistano sezioni di pagine e pagine in ciascuna sezione.

Le sezioni corrispodono in numero alle entry della tabella di primo livello, essendo queste 512 abbiamo che in un indirizzo 9 bit (i piu' significativi) si usano per indicare il numero di sezione di pagine a cui si accede.

Una volta deteminata la sezione, la relativa tabella delle pagine ha 512 entry, per cui nell'indirizzo 9 bit (i secondi piu' significativi) sono usati per identificare l'effettiva pagina acceduta.

Il resto dei 40 bit (ovvero 40 – 9 – 9) sono usati per l'offset all'interno della pagina acceduta. L'indirizzo ha quindi una struttura 9-9-22.

Il numero massimo di frame impegnati dalle pagine del processo dipende dal numero di tabelle di secondo livello attualmente utilizzate. Supponendo di avere 10 tabelle utilizzate in tutto, una sara' di primo livello e ne resteranno 9 di secondo livello, ciascuna delle quali puo' portarci a usare (al piu') 512 frame, per un totale di memoria (frame) massimo utilizzabile di 9 x 512.

PROVA D’ESAME DEL 13/6/2022

Si descriva lo scheduling di CPU round-robin virtuale. Si consideri uno scenario in cui lo scheduler di CPU round-robin virtuale sia basato su un time-slice pari a K > 10 millisecondi, esista un unico processo CPU-bound A di durata infinita ed esista un unico processo I/O-bound B di durata infinita.

Il processo B ha CPU burst di lunghezza pari a 10 millisecondi. Si indichi il massimo valore di K che possa permettere al processo B di avere un tempo di attesa per l’esecuzione del CPU burst non più ampio del 250% rispetto alla durata del time-slice.

Non ci sono ipotesi sulla durata della permanenza del processo B nello stato blocked.

Qiundi il processo B puo’ rientrare nello stato ready in un momento qualsiasi lungo il wall-clock-time.

Quando rientra nello stato ready puo’ dover attendere per prendere la CPU tutto un quanto di tempo K.

Allo stesso tempo e’ possibile che quando B riprende il controllo della CPU esso lo faccia nella coda di priorita’ HIGH (non in quella REGULAR), per cui e’ anche possibile che non riesca a completare il CPU-burst e venga mandato in preemption.

In tale scenario, otteniamo che B ripassa in coda REGULAR, e necessitera’ di attendere un intero quanto K.

Il tempo di attesa potra’ quindi essere al piu’ 2 x K, per cui 2 x K <= 250% di 10 millisecondi, ovvero K <= 25/2 millisecondi