

Elementi di Architettura e Sistemi Operativi

Bioinformatica - Tiziano Villa

17 Giugno 2013

Nome e Cognome:

Matricola:

Posta elettronica:

problema	punti massimi	i tuoi punti
problema 1	7	
problema 2	7	
problema 3	6	
problema 4	10	
totale	30	

1. Si consideri il seguente codice incompleto per scrivere le primitive di un semaforo P (*Wait*) and V (*Signal*) per un sistema uniprocessore.

```
typedef struct {
    int count;
} semaphore;

P(semaphore *S) {
    while (1) {
        disabilita le interruzioni;
        if (S->count ? ) {
            S->count ?;
            abilita le interruzioni
            return;
        }
        else abilita le interruzioni;
    }
}

V(semaphore *S) {
    disabilita le interruzioni;
    S->count ?;
    abilita le interruzioni
}
```

- (a) Si descriva brevemente che cos'è un semaforo e si mostri lo pseudo-codice della definizione classica delle operazioni P e V .

Traccia di soluzione.

Un semaforo è una variabile intera cui si può accedere, escludendo l'inizializzazione, solo tramite due operazioni atomiche predefinite: P (*Wait*) and V (*Signal*).

Le definizioni classiche di *Wait* e *Signal* in pseudo-codice sono le seguenti:

```
Wait (S)  {  
    while (S <= 0)  
        ;  
    S--;  
}
```

```
Signal (S) {  
    S++;  
}
```

- (b) Si completi il codice iniziale sostituendo i "?" con le parti mancanti, e lo si commenti spiegando perché realizza correttamente le primitive di un semaforo.

Traccia di soluzione.

La disabilitazione e riabilitazione delle interruzioni sono sufficienti per ottenere l'atomicità delle operazioni in caso di processore singolo.

```
typedef struct {
    int count;
} semaphore;

P(semaphore *S) {
    while (1) {
        disabilita le interruzioni;
        if (S->count > 0 ) {
            S->count -= 1;
            abilita le interruzioni
            return;
        }
        else abilita le interruzioni;
    }
}

V(semaphore *S) {
    disabilita le interruzioni;
    S->count += 1;
    abilita le interruzioni
}
```

(c) Che problemi ha il codice precedente ?

Traccia di soluzione.

Attesa attiva in P (per il ciclo $while(1)$).

Non e' equo in P .

(Stallo in P per certe priorit').

(d) Si modifichi il codice come segue: si aggiunga al semaforo una coda di processi in attesa; se P fallisce si aggiunga il processo alla coda, con V si tolga il processo dalla coda se essa non e' vuota.

Si commenti la soluzione risultante rispetto a quella iniziale e ai problemi del punto precedente.

Traccia di soluzione.

```
typedef struct {
    int count;
    queue q;
} semaphore;

P(semaphore *S) {
    disabilita le interruzioni;
    if (S->count > 0) {
        S->count -= 1;
        abilita le interruzioni
        return;
    }
    else {
        aggiungi il processo alla coda S->q;
        abilita le interruzioni;
    }
}

V(semaphore *S) {
    disabilita le interruzioni;
    S->count += 1;
    if (la coda S->q non e' vuota) {
        risveglia il primo processo di S->q
    }
    abilita le interruzioni
}
```

Questa soluzione migliora quella iniziale perche' l'introduzione della coda permette di risolvere i problemi del punto precedente, ad es. non presenta attesa attiva.

2. Si consideri il seguente algoritmo di schedulazione con diritto di prelazione, e basato su priorit  variabili dinamicamente. I numeri di priorit  maggiori indicano una priorit  pi  alta. Quando un processo   in attesa del processore (nella coda dei processi pronti), la sua priorit  varia a un tasso α ; quando   in esecuzione, la sua priorit  varia a un tasso β . All'ingresso nella coda dei processi pronti, si attribuisce la priorit  0 a tutti i processi. I parametri α e β si possono impostare in modo da fornire algoritmi di schedulazione diversi.
- (a) Si descriva brevemente che cosa vuol dire schedulazione con diritto di prelazione.
 - (b) Si descriva l'algoritmo risultante da $\beta > \alpha > 0$.
 - (c) Si descriva l'algoritmo risultante da $\alpha < \beta < 0$.
 - (d) Si confrontino brevemente i pro e i contro dei due algoritmi.

Traccia per lo svolgimento.

Per rispondere ai quesiti b) e c), si simuli l'arrivo di 3 processi nella coda dei pronti con i due scenari contraddistinti dalle relazioni tra α e β indicate. Per il caso c) si ponga attenzione al fatto che i tassi di cambiamento delle priorit  sono negativi.

Traccia di soluzione.

Caso $\beta > \alpha > 0$. Il processo in esecuzione aumenta la sua priorita' piu' velocemente dei processi in attesa.

Il primo processo che arriva nella coda dei pronti *processo1* ha la priorita' definita a 0, acquisisce il processore, e poi incrementa la sua priorita' piu' in fretta degli altri processi *processo2* *processo3* ... che arrivassero nella coda dei pronti (anch'essi partirebbero da 0 e poi aumenterebbero la priorita' a un tasso $0 < \alpha < \beta$). Percio' il *processo1* continuerebbe a detenere il processore, essendo la sua priorita' sempre maggiore di quella dei processi *processo2*, *processo3*

In definitiva questa e' la politica di schedulazione FIFO (detta anche FCFS), cioe' chi arriva per primo nella coda dei pronti e' servito per primo.

Caso $\alpha < \beta < 0$. I processi in attesa decrementano la loro priorita' piu' velocemente del processo in esecuzione.

Il primo processo che arriva nella coda dei pronti *processo1* ha la priorita' definita a 0, acquisisce il processore, e poi decrementa la sua priorita' al tasso $\beta < 0$. Supponiamo ora che arrivi il processo *processo2*, ad esso sarebbe assegnata la priorita' $0 > \beta$ e quindi il *processo2* si prenderebbe il processore interrompendo l'esecuzione del processo *processo1* che sarebbe rimesso nella coda dei pronti. Non solo, il processo *processo2* continuerebbe a tenere il processore, perche' la sua priorita' diminuirebbe con tasso β meno in fretta di quella del processo *processo1* che diminuirebbe con tasso α (dato che $\alpha < \beta < 0$). Se poi arrivasse un terzo processo *processo3*, esso avrebbe la priorita' definita a 0 e percio' si prenderebbe il processore alle spese del processo *processo2* e continuerebbe a tenere il processore finche' gi servisse, mentre i processi *processo1* e *processo2* rimarrebbero nella coda dei pronti. Quando il processo *processo3* non avesse piu' bisogno del processore, sarebbe ripescato dalla coda dei pronti il processo *processo2* e solo al suo termine sarebbe ripreso il processo *processo1*.

In definitiva questa e' la politica di schedulazione LIFO, cioe' chi arriva per ultimo nella coda e' servito per primo.

FIFO: i lavori sono eseguiti nell'ordine di arrivo e non si esercita la prelazione; il tempo di risposta medio puo' essere grande se lavori brevi devono aspettare dietro quelli lunghi (effetto a convoglio). Puo' portare a sovrapposizione poco efficiente dell'uso del processore e delle operazioni di ingresso/uscita.

LIFO: migliora il tempo di risposta dei nuovi processi, ma può precludere all'infinito il processore ai processi vecchi (affamare, "starvation").

3. Si consideri uno schema di traduzione da indirizzi logici a indirizzi fisici basato sull'impaginazione semplice.

Le pagine abbiano 4 bytes ciascuna. Siano date tre pagine della memoria logica, con indirizzo logico rispettivo: 0x00, 0x04 e 0x08.

La tavola delle pagine abbia tre elementi con i seguenti contenuti: l'elemento d'indice 0 ha contenuto 4 (in binario 000100), l'elemento d'indice 1 ha contenuto 3 (in binario 000011), l'elemento d'indice 2 ha contenuto 1 (in binario 000001).

- (a) Si mostri graficamente lo schema d'indirizzamento.

(b) Si calcolino gl'indirizzi delle pagine fisiche corrispondenti alle pagine logiche indicate, in particolare si calcolino:

- i. l'indirizzo fisico del primo elemento della pagina logica d'indirizzo $0x00$,
- ii. l'indirizzo fisico del primo elemento della pagina logica d'indirizzo $0x04$,
- iii. l'indirizzo fisico del primo elemento della pagina logica d'indirizzo $0x08$.

Traccia di soluzione.

- i. indirizzo pagina logica $0x00 = 0000_2$
 $0x00 = 00000000$, le ultime due cifre sono lo spiazzamento 00 le prime 6 cifre indicano l'elemento 0 della tavola delle pagine che contiene $4 = 0100_2$
concatenando 0100 con 00 si ha $010000_2 = 16_{10} = 10_{16}(0x10)$ indirizzo pagina fisica
- ii. indirizzo pagina logica $0x04 = 0100_2$
 $0x04 = 00000100$, le ultime due cifre sono lo spiazzamento 00, le prime 6 cifre indicano l'elemento 1 della tavola delle pagine che contiene $3 = 0011_2$
concatenando 0011 con 00 si ha $001100_2 = 12_{10} = C_{16}(0xC)$ indirizzo pagina fisica
- iii. indirizzo pagina logica $0x08 = 1000_2$
 $0x08 = 00001000$, le ultime due cifre sono lo spiazzamento 00, le prime 6 cifre indicano l'elemento 2 della tavola delle pagine che contiene $1 = 0001_2$
concatenando 0001 con 00 si ha $000100_2 = 4_{10} = 4_{16}(0x4)$ indirizzo pagina fisica

4. Si progetti un circuito sequenziale che realizza la seguente specifica:

- Ci sono un segnale binario d'ingresso X e un segnale binario d'uscita Z .
- L'uscita z vale 1 al tempo t se e solo se il valore assunto da x al tempo t e' identico a quello assunto da x a due unita' di tempo precedenti: $x(t) = x(t - 2)$. In tutte le altre condizioni z vale 0.

(a) Si disegni il grafo delle transizioni di una macchina a stati finiti di tipo Mealy che corrisponde alla specifica. S'indichi lo stato iniziale.

Nota. Si supponga che il circuito sia stato portato nello stato iniziale con la sequenza d'ingressi 000, cioe' che quale che sia lo stato al momento dell'accensione al circuito sia applicata la sequenza d'ingressi 000 per portarlo nello stato iniziale. Ne consegue in particolare che nello stato iniziale il circuito produrra' 1 sotto l'ingresso 0 (corrispondente alla sequenza d'ingressi piu' recente 000) e produrra' 0 sotto l'ingresso 1 (corrispondente alla sequenza d'ingressi piu' recente 001).

Traccia di soluzione.

Si veda il libro di testo Sezione 6.8, Esempio 1).

- (b) Si minimizzi il numero degli stati della macchina proposta, applicando l'algoritmo di minimizzazione degli stati.

- (c) Si scriva la tavola delle transizioni con gli stati futuri e le uscite e la si codifichi.

- (d) Supponendo di usare bistabili di tipo D, si derivino le equazioni minimizzate di eccitazione degl'ingressi dei bistabili e le equazioni minimizzate delle uscite.

- (e) Si realizzi il circuito sequenziale corrispondente con bistabili di tipo D campionati sul fronte di salita, invertitori e porte NAND. Si etichettino con chiarezza i segnali.