

Elementi di Architettura e Sistemi Operativi

Bioinformatica - Tiziano Villa

16 Luglio 2018

Nome e Cognome:

Matricola:

Posta elettronica:

problema	punti massimi	i tuoi punti
problema 1	7	
problema 2	8	
problema 2	5	
problema 3	10	
totale	30	

1. Si consideri il problema di sincronizzazione cosiddetto degli n filosofi, dove n filosofi siedono attorno a una tavola con una bacchetta tra ogni coppia di filosofi vicini.

I filosofi sono numerati da 0 a $n - 1$ e ad essi corrispondono processi diversi, cioè' ogni filosofo esegue $Pranza(i)$, dove i e' il numero del filosofo. Si assuma che ci sia un vettore di semafori, $Bacchetta[i]$ che rappresenta la bacchetta alla sinistra del filosofo i . Tutti i semafori sono inizializzati a 1.

Si consideri la seguente soluzione:

```
void Pranza(int i) {  
    Bacchetta[i].P();    /* prendi la bacchetta sinistra */  
    Bacchetta[(i+1)%n].P(); /* prendi la bacchetta destra */  
    Mangia();  
    Bacchetta[i].V();    /* lascia la bacchetta sinistra */  
    Bacchetta[(i+1)%n].V(); /* lascia la bacchetta destra */  
}
```

- (a) Si spieghi come funziona questo codice di sincronizzazione.

Traccia di soluzione.

I semafori garantiscono la sincronizzazione nell'accesso alle bacchette.

Si veda il libro di testo.

- (b) Questa soluzione soddisfa le condizioni necessarie per lo stallo ? Si elenchino le condizioni di stallo, e per ogni condizione di stallo si verifichi se essa e' soddisfatta oppure o no in questa soluzione.

Traccia di soluzione.

- I semafori sono inizializzati a 1, percio' ogni bacchetta puo' essere detenuta da un solo processo per volta.
- Niente prelazione. Le bacchette non possono essere sottratte a chi le detiene senza violare la semantica dei semafori su cui si basa il codice precedente.
- Possesso e attesa.
In uno stallo, la seconda $P()$ nel codice proposto ha come effetto che il processo attenda mentre detiene la prima bacchetta ottenuta con la prima $P()$.
- Attesa circolare.
Il filosofo i afferra la $Bacchetta[i]$ e aspetta che il filosofo $(i + 1) \% n$ rilasci la $Bacchetta[(i + 1) \% n]$.
C'e' un ciclo chiuso dal filosofo $n - 1$ quando afferra la $Bacchetta[n - 1]$ e aspetta che il filosofo 0 rilasci la $Bacchetta[0]$.

- (c) Questa soluzione puo' entrare in stallo ? Se no, si argomenti perche' no. Se si, si mostri una successione di chiamate del processo *Pranza()* (indicando il relativo argomento) che porta allo stallo.

Traccia di soluzione.

Si, puo' entrare in stallo perche' le condizioni necessarie sono soddisfatte, come visto al punto precedente.

Si ha uno stallo quando si esegue *Pranza(0)* e tale chiamata e' sospesa dopo la prima *P()*, poi si esegue *Pranza(1)* e tale chiamata e' sospesa dopo la prima *P()*, poi si esegue *Pranza(2)* e tale chiamata e' sospesa dopo la prima *P()*, poi si esegue *Pranza(3)* e tale chiamata e' sospesa dopo la prima *P()*, poi si esegue *Pranza(4)* e tale chiamata e' sospesa dopo la prima *P()*. A questo punto ogni filosofo ha afferrato la sua bacchetta sinistra, ma non ci sono altre bacchette libere, tutti i filosofi rimangono bloccati sulla seconda *P()* in attesa che un altro filosofo rilasci la sua bacchetta, senza che nessuno possa mangiare e quindi rilasciare le bacchette detenute.

- (d) Si consideri la seguente variante della soluzione iniziale (dove *lucchetto* e' una variabile globale condivisa da tutti i processi).

```
Lucchetto lucchetto;
void Pranza(int i) {
    lucchetto.Acquisisci();
    Bacchetta[i].P(); /* prendi la bacchetta sin. */
    Bacchetta[(i+1)%n].P(); /* prendi la bacchetta des. */
    lucchetto.Rilascia();
    Mangia();
    Bacchetta[i].V(); /* lascia la bacchetta sin. */
    Bacchetta[(i+1)%n].V(); /* lascia la bacchetta des. */
}
```

Con questa soluzione puo' darsi uno stallo ? Se si, si mostri una successione di chiamate che porta allo stallo. Se no, si discuta quale condizione necessaria di stallo e' invalidata. Si spieghi come funziona questa soluzione.

Traccia di soluzione

Questa soluzione usa un lucchetto per assicurarsi che ad un filosofo siano assegnate entrambe le bacchette, percio' non piu' di un filosofo puo' essere bloccato nella sezione critica in attesa di prendere entrambe le bacchette. I filosofi rimanenti o avranno due bacchette (e potranno mangiare) o non avranno alcuna bacchetta e saranno in attesa che l'attuale detentore del lucchetto lo rilasci. Quindi s'invalida la condizione necessaria di attesa circolare, perche' il grafo risultante e' aciclico.

- (e) Si consideri la seguente variante della soluzione iniziale (dove *lucchetto* e *InUso* sono variabili globali condivise da tutti i processi).

```
Lucchetto lucchetto;
Boolean[n] InUso;
void Pranza(int i) {
    Boolean successo = falso;
    while (!successo) {
        lucchetto.Acquisisci();
        if (!InUso[i] && (!InUso[(i+1)%n])) {
            Bacchetta[i].P(); /* prendi la bacchetta sin. */
            Bacchetta[(i+1)%n].P(); /* prendi la bacchetta des. */
            InUso[i] = vero;
            InUso[(i+1)%n] = vero;
            successo = vero;
        } else {
            Cedi_Processore();
        }
        lucchetto.Rilascia();
    }
    Mangia();
    Bacchetta[i].V(); /* lascia la bacchetta sin. */
    Bacchetta[(i+1)%n].V(); /* lascia la bacchetta des. */
    InUso[i] = falso;
    InUso[(i+1)%n] = falso;
}
```

Con questa soluzione puo' darsi uno stallo ? Se si, si mostri una successione di chiamate che porta allo stallo. Se no, si discuta quale condizione necessaria di stallo e' invalidata. Si spieghi come funziona questa soluzione. [Non si discuta la semantica della cessione volontaria del processore rispetto al possesso del lucchetto]

Traccia di soluzione.

Si noti che il vettore *InUse* e' inizializzato a *falso* (0) dal compilatore (se ci sono dubbi al riguardo, meglio inizializzarlo esplicitamente a *falso*).

Questa soluzione invalida la condizione necessaria di possesso e attesa

perche' o un filosofo prende entrambe le bacchette se sono libere oppure cede volontariamente il processore (per ritornare a richiedere le bacchette quando riattivato).

Si rifletta sulla diversita' delle due soluzioni proposte. Nel punto precedente un solo filosofo puo' possedere una bacchetta ed essere in attesa di un'altra (quindi s'invalida l'attesa circolare, ma resta il possesso ed attesa), in questa soluzione un filosofo o prende due bacchette o nessuna.(quindi s'invalida il possesso e attesa, e ovviamente anche l'attesa circolare come conseguenza del non esserci piu' possesso e attesa).

[Si puo' pensare che all'atto della cessione volontario si rilasci il lucchetto e lo si riacquisisca all'atto della riattivazione, ma questa parte e' stata volutamente esclusa dalla discussione della presente domanda]

2. Si consideri un sistema di paginazione su richiesta con un disco di paginazione che abbia un tempo medio di accesso e trasferimento di 20 milisecondi. Gli indirizzi siano tradotti per mezzo di una tabella delle pagine che si trova in memoria centrale con un tempo di accesso di un microsecondo per ogni accesso alla memoria, per cui ogni riferimento alla memoria per mezzo della tavola delle pagine richiede due accessi alla memoria (uno per l'indirizzo e uno per il dato). Per migliorare questo tempo s'introduce una memoria associativa delle traduzioni degli indirizzi (TLB) che riduce il tempo di accesso a un solo riferimento alla memoria, se l'elemento della tabella delle pagine si trova nella memoria associativa (se l'indirizzo si trova nella memoria associativa serve un solo accesso alla memoria centrale per il dato).

Supponendo che per l'80% degli accessi l'elemento della tabella delle pagine corrispondente (la traduzione dell'indirizzo) si trovi nella memoria associativa, e che il 10% dei restanti (cioè il 2% del totale) causi un'assenza di pagina, si calcoli il tempo effettivo (medio) di accesso alla memoria in μs .

Traccia di soluzione.

$$TEA = (0,8 \times 1\mu s) + (0,18 \times 2\mu s) + (0,02 \times (20.000\mu s + 2\mu s)) = 0,8\mu s + 0,36\mu s + 400,04\mu s = 401,2\mu s.$$

- (a) Il primo contributo è un riferimento al dato nella memoria centrale quando la traduzione dell'indirizzo si trova nella memoria associativa
- (b) Il secondo contributo sono due riferimenti alla memoria centrale, uno per trovare l'indirizzo che non è presente nella memoria associativa ma si ottiene dalla tabella nella memoria centrale, l'altro per il dato nella memoria centrale
- (c) Il terzo contributo è per portare una nuova pagina dal disco alla memoria centrale, più l'indirizzo che si trova nella tavola delle pagine nella memoria centrale, più il dato (adesso portato nella memoria centrale)
- (d) Il tempo della memoria associativa è trascurato

3. Dato il seguente frammento di programma nel linguaggio di LC-3

```
START    LDI      R1, A
          BRzpb    START
          STI      R0, B
          BRnzb    PROX
A         .FILL    xFE04
B         .FILL    xFE06
```

si spieghi che cosa fa e si commenti ogni singola istruzione.

Traccia di soluzione

Nelle prime due righe s'interroga ciclicamente $DSR[15]$ per vedere se si e' finito con l'ultimo carattere inviato dal processore. Se $DSR[15] = 0$ si sta ancora processando il carattere e si ripete un ciclo saltando a *START*. Quando il circuito dello schermo ha processato il carattere, automaticamente assegna $DSR[15] = 1$, per cui si procede all'istruzione *STI* che memorizza il contenuto di *R0* all'indirizzo $xFE06$, l'indirizzo mappato sulla memoria del registro *DDR*. La scrittura di *DDR* azzerava anche $DSR[15]$ disabilitando per il momento il registro *DDR* dalla gestione di un altro carattere in uscita. Il carattere in *DDR* e' reso visibile sullo schermo dal circuito dello schermo stesso. Infine si salta incondizionatamente a *PROX*. Si notino la lettura indiretta *LDI* a $xFE04$ (indirizzo mappato sulla memoria del registro *DSR*) e la scrittura indiretta *STI* a $xFE06$ (indirizzo mappato sulla memoria del registro *DDR*).

Si veda il materiale sulla gestione dell'uscita nell'architettura LC-3.

4. Si progetti un sommatore binario combinatorio per due operandi con 4 cifre binarie, ottenuto combinando moduli sommatore per due operandi con una cifra binaria. Siano A e B gl'ingressi, $S = A + B$ il risultato, RI il riporto in ingresso, RU il riporto in uscita.

Si proceda come segue:

- (a) Si mostri la tavola di verita' del semi-addizionatore binario a una cifra avente in ingresso due operandi A e B e in uscita la somma $S = A + B$ e il riporto in uscita.
- (b) Si mostri la tavola di verita' del sommatore completo che in ingresso ha anche RI il riporto in ingresso.
- (c) Si minimizzi la logica risultante a due livelli.
- (d) Si mostri una realizzazione circuitale della logica ottenuta al punto precedente.
- (e) Si mostri come combinare piu' unita' del precedente sommatore a una cifra binaria per ottenere un sommatore su quattro cifre binarie.

Traccia di soluzione

Si consultino il libro di testo e/o le disponse.