Elementi di Architettura e Sistemi Operativi

Bioinformatica - Tiziano Villa

2 Luglio 2018

Nome e Cognome:
Matricola:
Posta elettronica:

problema	punti massimi	i tuoi punti
problema 1	15	
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, cioe' 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.

- Mutua esclusione.
  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.

```
void Pranza(int i) {
 if (i == n-1) {
  Bacchetta[?].P();   /* ? */
  Bacchetta[?].P();/* ? */
 } else {
  Bacchetta[i].P();   /* prendi la bacchetta sin. */
  Bacchetta[(i+1)%n].P();/* prendi la bacchetta des. */
 }
 Mangia();
 Bacchetta[i].V();   /* lascia la bacchetta sin. */
 Bacchetta[(i+1)%n].V();/* lascia la bacchetta des. */
}
```

Si completi la soluzione in modo che sia invalidata una delle condizioni necessarie di stallo (si completi dove ci sono dei punti di domanda). Quale condizione di stallo e' invalidata ? Si spieghi come funziona questa soluzione.

Traccia di soluzione

Nella soluzione originale le risorse sono richieste sempre nell'ordine prima quella a sinistra poi quella a destra e questo puo' causare lo stallo indicato prima. Si puo' eliminare l'attesa circolare imponendo che il filosofo d'indice 0 richieda prima la bacchetta a destra poi quella a sinistra. Considerando gl'indici delle bacchette, nella soluzione originale il filosofo d'indice 0 prende le bacchette nell'ordine a sinistra indice 0 e a destra indice 1; il filosofo d'indice 1 prende le bacchette nell'ordine a sinistra indice 1 e a destra indice 2; ...; il filosofo d'indice $n-1$ prende le bacchette nell'ordine a sinistra indice $n-1$ e a destra indice 0, in altri termini questo filosofo non prende le bacchette in ordine crescente come gli altri.

Nella nuova soluzione, il (solo) filosofo d'indice $n-1$ prende le bacchette nell'ordine a destra indice 0 e a sinistra indice $n-1$, quindi anch'egli prende le bacchette in ordine d'indice crescente. In questo modo le risorse sono richieste in ordine sempre crescente rispetto all'indice, e percio' non c'e' piu' attesa circolare, la quale richiede che un filosofo prenda le bacchette nell'ordine decrescente opposto agli altri che le prendono in ordine crescete.

```
void Pranza(int i) {
 if (i == n-1) {
  Bacchetta[0].P();   /* prendi la bacchetta des. */
  Bacchetta[n-1].P();/* prendi la bacchetta sin. */
 } else {
  Bacchetta[i].P();   /* prendi la bacchetta sin. */
  Bacchetta[(i+1)%n].P();/* prendi la bacchetta des. */
 }
 Mangia();
 Bacchetta[i].V();   /* lascia la bacchetta sin. */
 Bacchetta[(i+1)%n].V();/* lascia la bacchetta des. */
}
```

Si noti che e' la medesima cosa scrivere

```
 if (i == n-1) {
  Bacchetta[0].P();   /* prendi la bacchetta des. */
  Bacchetta[n-1].P();/* prendi la bacchetta sin. */
 } else {
```

oppure

```
 if (i == n-1) {
  Bacchetta[(i+1)%n].P();   /* prendi la bacchetta des. *
  Bacchetta[i].P();/* prendi la bacchetta sin. */
 } else {
```

(e) Si consideri la seguente variante della soluzione iniziale.

```
void Pranza(int i) {
 if (i%2 == 1) {
  Bacchetta[i].P();   /* prendi la bacchetta sin. */
  Bacchetta[(i+1)%n].P();/* prendi la bacchetta des. */
 } else {
  Bacchetta[?].P();/* ? */
  Bacchetta[?].P();   /* ? */
 }
 Mangia();
 Bacchetta[i].V();   /* lascia la bacchetta sin. */
 Bacchetta[(i+1)%n].V();/* lascia la bacchetta des. */
}
```

Si completi la soluzione in modo che sia invalidata una delle condizioni necessarie di stallo (si completi dove ci sono dei punti di domanda). Quale condizione di stallo e' invalidata ? Si spieghi come funziona questa soluzione.

Traccia di soluzione

I filosofi d'indice dispari prendono le bacchette nell'ordine sinistra e destra, quelli d'indice pari le prendono nell'ordine destra e sinistra. Un blocco richiede che tutti i filosofi abbiano una bacchetta ma siano in attesa circolare dell'altra. Con questa nuova soluzione non puo' accadere che una coppia di filosofi adiacenti (per forza uno pari e uno dispari) siano in attesa della bacchetta tra loro perche' in questo schema uno di loro prendera' la bacchetta e potra' mangiare.

```
void Pranza(int i) {
 if (i%2 == 1) {
  Bacchetta[i].P();   /* prendi la bacchetta sin. */
  Bacchetta[(i+1)%n].P();/* prendi la bacchetta des. */
 } else {
  Bacchetta[(i+1)%n].P();/* prendi la bacchetta des. */
  Bacchetta[i].P();   /* prendi la bacchetta sin. */
 }
 Mangia();
 Bacchetta[i].V();   /* lascia la bacchetta sin. */
```

```
 Bacchetta[(i+1)%n].V();/* lascia la bacchetta des. */
}
```

Non e' corretto rispondere che questa nuova soluzione invalida la condizione necessaria chiamata "possesso e attesa" perche' con questa soluzione rimangono delle situazioni in cui un processo puo' possedere una bacchetta e aspettare l'altra, ma in cui l'attesa non e' circolare.

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

```
START   LDI    R1, A
        BRzp   START
        LDI    R0, B
        BRnzp  PROX
A       .FILL  xFE00
B       .FILL  xFE02
```

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

Traccia di soluzione

Carica nel registro $R0$ il codice ASCII che e' stato immesso da tastiera e poi salta all'istruzione (non visibile) indirizzata dall'etichetta $PROX$.

Per leggere si usano i registri $KBDR$ (registro dei dati) e $KBSR$ (registro di sincronizzazione) cui sono assegnati rispettivamente gl'indirizzi $xFE00$ e $xFE02$ dallo spazio d'indirizzi della memoria (ingresso mappato sulla memoria).

Quando si pigia un tasto, il codice ASCII corrispondente e' caricato in $KBDR[7:0]$ e $KBSR[15]$ e' messo a 1. Quando un'istruzione di LC-3 legge il carattere in $KBDR$, $KBSR[15]$ e' azzerato per permettere di pigiare un altro tasto e cosi' leggere un altro carattere. Se $KBSR[15]$ vale 1, il carattere immesso da tastiera non e' ancora stato letto nel registro $KBDR$ e percio' la tastiera e' disabilitata. Fino a quando $KBSR[15]$ vale 0, nessun tasto e' stato pigiato dall'ultima volta che un carattere e' stato letto in $KBDR$.

Applicando il meccanismo dell'interrogazione, il programma continua a controllare se $KBSR[15]$ (caricato nel registro $R1$ da LDI) vale 1 (cioe' se $R1$ e' negativo), e quando e' cosi' copia nel registro $R0$ il codice ASCII che si trova in $KBDR$.

Si usa l'istruzione di lettura indiretta LDI per caricare in un registro il contenuto della memoria d'indirizzo $xFE00$ e $XFE02$ (ad es., $LDI\ R1, A$ corrisponde a $R1 \leftarrow mem[mem[A]] = mem[xFE00]$).

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

3. Si progetti un sottrattore binario combinatorio per due operandi con 4 cifre binarie, ottenuto combinando moduli sottrattori per due operandi con una cifra binaria (sulla falsariga del sommatore binario combinatorio), Siano A e B gl'ingressi, $D = A - B$ il risultato, PD il prestito concesso a destra, PS il prestito richiesto a sinistra.

   Si proceda come segue:

   (a) Si mostri la tavola di verita' del semi-sottrattore binario a una cifra avente in ingresso due operandi A e B e in uscita la differenza $D = A - B$ e il prestito richiesta a sinistra.

   (b) Si mostri la tavola di verita' del sottrattore completo che in ingresso ha anche PD il prestito concesso a destra.

   (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 sottrattore a una cifra binaria per ottenere un sottrattore su quattro cifre binarie.

   Traccia di soluzione

   Semi-sottrattore

A	B	D	PS
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

   Sottrattore totale

A	B	PD	D	PS
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1