

Sistemi Operativi

Esercizi Ricapitolazione

Docente: Claudio E. Palazzi
cpalazzi@math.unipd.it

Problema

- Numerosi operai in una fabbrica preparano un unico prodotto con l'utilizzo di **nA** quantità del componente **A**, **nB** quantità del componente **B**. Un fattorino viene chiamato a riempire le quantità di **A** e **B** fino a **totA** e **totB** ogni volta che le loro quantità residue scendono sotto **nA** ed **nB**. Prima di iniziare a comporre il prodotto, ogni operaio si assicura di avere le quantità necessarie dei due componenti; viceversa chiama il fattorino e attende che arrivi con la scorta. Una volta terminato il suo compito il fattorino resta in attesa di essere richiamato. Una volta completato un prodotto ogni operaio inizia a prepararne un altro. Inizialmente le scorte di **A** e **B** sono piene e nessun prodotto è stato ancora preparato.
- Scrivere le seguenti procedure che propongono una soluzione a questo problema utilizzando i costrutti dei monitor.

Soluzione

Il monitor potrebbe utilizzare i seguenti dati:

operaio: variabile condition sulla quale gli operai si sospendono in attesa di essere riforniti di un determinato componente;

fattorino: variabile condition sulla quale il fattorino si sospende in attesa di essere mandato a comprare le scorte di un componente;

operai_in_attesa: intero che indica il numero degli operai in attesa di un componente;

nA, nB: costanti che indicano la quantità necessaria di ogni componente per la preparazione di un prodotto;

totA, totB: costanti che indicano il numero massimo di scorte per ogni componente;

qA, qB: interi che indicano la quantità di componenti **A** e **B** attualmente presenti in fabbrica.

```
monitor Fabbrica {  
    condition operaio, fattorino;  
    int qA, qB;  
    int operai_in_attesa;
```

Soluzione

Funzione invocata da ciascun operaio all'infinito

```
void prepara_prodotto {  
    while (true) {  
        if (qA > nA) and (qB > nB) {  
            qA = qA - nA;  
            qB = qB - nB;  
        }  
        else {  
            fattorino.signal();  
            operai_in_attesa++;  
            operaio.wait();  
        }  
    }  
}
```

Soluzione

Funzione invocata da fattorino

```
void scorta_ingrediente {  
    qA = totA;  
    qB = totB;  
    while (operai_in_attesa > 0 ) {  
        operaio.signal();  
        operai_in_attesa --;  
    }  
    fattorino.wait();  
}
```

Problema

- Sincronizzazione di 3 Processi coi Semafori.
- Si considerino i processi A, B e C che si sincronizzano come mostrato nel seguito attraverso i semafori Sem1, Sem2 e Sem3 (inizializzati a 0) e che operano sulle variabili condivise x, y e z, che sono inizializzate come segue:
 $x = 1; y = 2; z = 1.$
- Con quale ordine i processi stampano i valori delle tre variabili?
- Qual è il valore delle tre variabili che viene infine stampato?

Problema

Inizializzaz.

$x = 1;$

$y = 2;$

$z = 1;$

$Sem1 = 0;$

$Sem2 = 0;$

$Sem3 = 0;$

Process A {

P(Sem1);

$z = (x - z) * y;$

$x = x + z + y;$

V(Sem3);

P(Sem1);

$x = x + y;$

Print(x);

}

Process B {

P(sem2);

$x = x + y;$

$z = x - z;$

$y = (y - z) + x;$

Print(y);

V(sem1);

}

Process C {

V(Sem2);

P(Sem3);

$x = x / y;$

$y = 2z + x;$

V(Sem1);

$z = x + z;$

Print(z);

}

Soluzione

- Il processo B stampa la variabile y per primo.
- Non possiamo fare affermazioni su quale processo stampa per secondo e per terzo tra A e C.
- Il risultato finale sarà $x = 12$ e $y = 3$ ma non possiamo dire se sarà $z = 6$ oppure $z = 15$ (a seconda di quale processo stampa per primo tra A e C)

Problema

- Si consideri una variante FAT-15 dell'architettura di file system nota come FAT-16, nella quale l'unica differenza dalla versione base sia che 1 bit dell'indice FAT non sia utilizzabile, tutti gli altri attributi di architettura rimanendo inalterati. Si specifichi la dimensione massima di file possibile con tale variante, e la dimensione massima di partizione.

Soluzione

- Come sappiamo l'indice X dell'architettura di file system FAT- X denota il numero di bit utilizzati per esprimere l'indice di blocco. Per le ipotesi del quesito abbiamo $X = 15$, il che significa che l'intera partizione potrà constare di $2^{15} = 32.768$ blocchi. Poiché il quesito richiede che tutte le altre caratteristiche dell'architettura in esame restino uguali allo standard FAT-16, i blocchi su disco saranno ampi al max $32 \text{ KB} = 2^5 \times 2^{10} \text{ B} = 2^{15} \text{ B}$. La dimensione massima di file così come la dimensione massima di partizione saranno dunque fissate a:
 $2^{15} \text{ blocchi} \times 2^{15} \text{ B / blocco} = 2^{30} \text{ B} = 1 \text{ GB}$,
ovvero meta' della dimensione max ottenibile con FAT-16.

Problema

In un sistema le pagine hanno una dimensione di 1kB (1024), e la RAM è suddivisa in 256 frame. Nelle page table, un numero di pagina è scritto su 2 byte, e la page table più grande ammessa dal sistema occupa completamente un frame della RAM (per semplicità non consideriamo dirty bit e bit di validità).

Rispondere alle seguenti domande:

- a) il sistema usa un algoritmo di rimpiazzamento delle pagine?
- b) qual è la lunghezza in bit di un indirizzo fisico?
- c) il fatto che in un generico sistema lo spazio di indirizzamento fisico sia più piccolo dello spazio di indirizzamento logico è condizione necessaria perché il sistema soffra del problema del *thrashing*?

Soluzione

- a) Si. Infatti una *page table* contiene al massimo $1024/2=512$ *entries*, mentre il numero di frame della RAM è 256.
- b) dimensione pagina= 2^{10} byte; numero di frame = $2^8 \Rightarrow$ indirizzo fisico di 18 bit
- c) No. Infatti può succedere che il sistema implementi la memoria virtuale e se la dimensione globale di tutti i processi attivi nel sistema eccede lo spazio di indirizzamento fisico allora è possibile il *thrashing*

Problema

In un sistema con paging, pagine di 2^6 bytes e la seguente page table

<i>In/Out</i>	<i>Frame</i>
in	00101
out	01011
out	00001
out	11010
in	00011
out	10101
out	11111
in	10101

dire se i seguenti indirizzi logici genereranno un *page fault*. In caso negativo, scrivere l'indirizzo fisico corrispondente:

a) 0000001001001

b) 0000011010110

c) 0000100000101

d) 0000000111100

Soluzione

Visto che la dimensione della pagina è 2^6 bytes, gli ultimi 6 bit dell'indirizzo virtuale rappresentano l'*offset*, mentre i rimanenti sono il numero di pagina.

a) 0000001 001001: pagina 1 (cominciando a contare da zero nella page table). *Page fault*.

b) 0000011 010110: pagina 3. *Page fault*.

c) 0000100 000101: pagina 4. Pagina valida.
Diventa: 00011000101

d) 0000000 111100: pagina 0. Pagina valida.
Diventa: 00101111100