Classici problemi di sincronizzazione

1. Bounded-Buffer Problem (PROD-CONS)
2. Readers and Writers Problem
3. Dining-Philosophers Problem

1. Bounded-Buffer Problem

Variabili condivise :

```
type item = …
var buffer = …
        full, empty, mutex: semaphore ;
        nextp, nextc: item ;
        full :=0; empty := n ; mutex :=1;
```

Semaforo per numero di locazioni piene

Semaforo per numero di locazioni vuote

Semaforo di mutua esclusione

1. Bounded-Buffer Problem

Processo produttore :

```
repeat
        …
    produci un elemento in nextp
        …
    wait(empty);
    wait(mutex);
        …
     inserisci nextp in buffer
        …
    signal(mutex);
    signal(full);
until false;
```


1. Bounded-Buffer Problem

Processo consumatore:

```
repeat
        wait(full)
        wait(mutex);
            …
        rimuovi l'elemento di buffer in nextc
            …
        signal(mutex);
        signal(empty);
            …
        consuma l'elemento in nextc
            …
until false;
```

Si noti la simmetria del codice con il produttore!

2. Readers and Writers Problem

PROBLEMA

Un'area di memoria deve essere ***condivisa*** tra vari processi che possono scrivere su (***writer***) o leggere da (***reader***) tale area

Piu' processi ***reader possono leggere contemporaneamente*** ma, per problemi di consistenza, un ***writer non puo' accedere*** all'area ***in contemporaneita'*** con nessun altro processo (reader o writer)

2. Readers and Writers Problem

SOLUZIONI

Prima versione – precedenza ai reader : nessun reader deve attendere a meno che un writer non sia gia' nella sezione critica

Seconda versione – precedenza ai writer : se un writer e' in attesa iniziera' al piu' presto, e cioe' quando i reader che stavano gia' dentro avranno terminato; gli altri reader attenderanno

Possibile starvation in entrambi i casi !!!

2. Readers and Writers Problem
(prima versione)

Variabili condivise :

var *mutex*, *wrt* : *semaphore* (=1);
readcount : *integer* (=0);

Processo writer :

wait(*wrt*);
...
esegue la scrittura
...
signal(*wrt*);

2. Readers and Writers Problem

Processo reader :

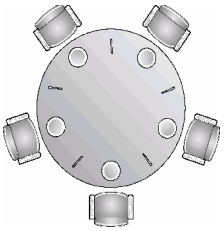
wait (*mutex*);
readcount := *readcount* +1;
if *readcount* = 1 **then** *wait* (*wrt*);
signal (*mutex*);
...
esegue la lettura
...
wait(*mutex*);
readcount := *readcount* – 1;
if *readcount* = 0 **then** *signal* (*wrt*);
signal(*mutex*);

Mentre un writer e' dentro il primo lettore si accoda su wrt mentre tutti i lettori seguenti all'ingresso del primo entrano direttamente

2. Readers and Writers Problem
(seconda versione)

CHE NE DITE DI PROVARCI DA SOLI, CHE VI FAREBBE TANTO BENE !?!?!?

3. Dining-Philosophers Problem



Ogni filosofo esegue questa sequenza:

- Pensa
- Si procura le due bacchette
- Mangia
- Ripone le posate

Variabili condivise :
var *chopstick* : **array** [0..4] **of** *semaphore* ;
(=1 inizialmente)

3. Dining-Philosophers Problem

Philosopher i :

repeat
wait (*chopstick*[*i*])
wait (*chopstick*[*i*+1 **mod** 5])
...
mangia
...
signal (*chopstick*[*i*]);
signal (*chopstick*[*i*+1 **mod** 5]);
...
pensa
...
until *false* ;

Soluzione inaccettabile : possibile deadlock! Ci ritorniamo…

Strutture ad alto livello per la gestione di sezioni critiche

L'uso dei ***semafori*** si presta comunque ad ***errori*** introdotti dal programmatore

⇩

Servono ***costrutti a piu' alto livello*** per gestire in maniera piu' sicura i problemi di sincronizzazione

1. Regioni critiche
2. Monitors

1. Regioni critiche: definizione

Una ***variabile condivisa*** *v* di tipo *T* viene dichiarata come:

var *v* : **shared** *T*

… si accede alla variabile *v* ***solo all'interno di questo tipo di istruzione***:

sezione critica

region *v* **when** *B* **do** *S*

dove *B* e' un'espressione booleana.

Mentre si esegue *S* ***nessun altro processo puo' accedere*** alla variabile *v*.

1. Regioni critiche: realizzazione

Quando un processo prova ad eseguire l'istruzione ***region***, l'espressione booleana ***B*** viene valutata

Se ***B*** e' vera, l'istruzione ***S*** viene eseguita se nessun altro processo e' in una regione associata con ***v***

Se ***B*** e' falsa, il processo e' sospeso finche' ***B*** diventa vera e nessun altro processo e' in una regione associata con ***v***

1. Regioni critiche: PROD-CONS

Variabili condivise :

```
var buffer :
      shared record
            pool : array [0..n–1] of item ;
            count,in,out: integer
      end;
```


Processo produttore :

```
region buffer when count < n
do begin
            pool [in] := nextp;
            (in := in +1) mod n ;
            count := count + 1;
      end;
```

Processo consumatore :

```
region buffer when count > 0
do begin
            nextc := pool [out];
            (out := out +1) mod n ;
            count := count – 1;
      end;
```