SEMAFORI

1

SEMAFORI

- Variabile intera non negativa con valore iniziale >= 0
- Al semaforo è associata una lista di attesa Qs nella quale sono posti i descrittori dei processi che attendono l'autorizzazione a procedere

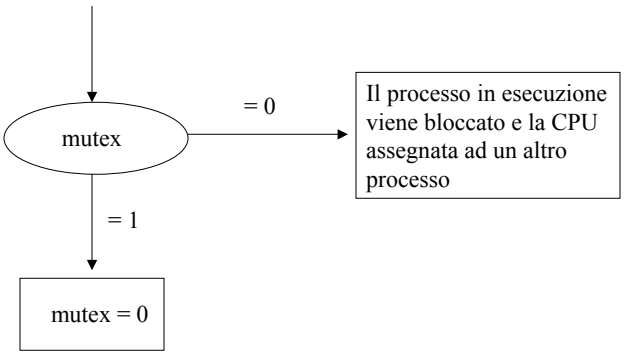
Sul semaforo sono ammesse solo due operazioni (primitive)

WAIT	SIGNAL
(P(s))	(V(s))

2

PRINCIPIO DI MUTUA ESCLUSIONE

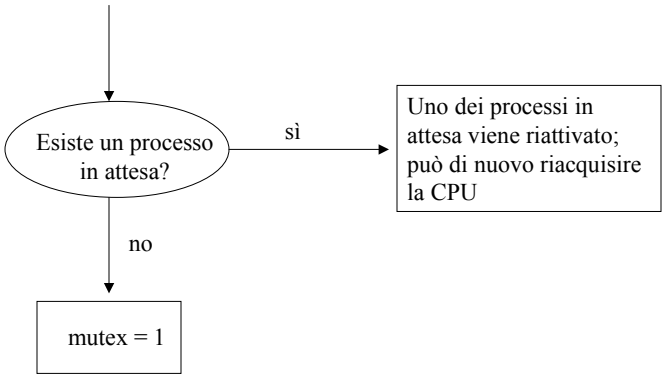
wait



3

PRINCIPIO DI MUTUA ESCLUSIONE

signal



4

wait(s):

if (s = 0) **then**

<il processo viene sospeso ed il suo descrittore inserito in Qs>

else

s := s + 1;

signal(s):

if (<esiste un processo in coda>) **then**

<il suo descrittore viene rimosso da Qs ed il suo stato modificato in *pronto*>

else

s := s + 1;

- L'esecuzione della signal non comporta concettualmente nesusna modifica nello stato del processo che l'ha eseguita.
- Scelta del processo tramite FIFO

5

- WAIT e SIGNAL devono essere azioni indivisibili (azioni atomiche)
- Analisi e modifica del valore del semaforo ed eventuale sospensione o riattivazione di un processo devono avvenire in modo indivisibile
- Durante un'operazione sul semaforo (WAIT e SIGNAL), nessun altro processo può accedere al semaforo fino a che l'operazione è completata o bloccata
- **WAIT e SIGNAL**: sezioni critiche

6

SOLUZIONE AL PROBLEMA DELLA MUTUA ESCLUSIONE

P1	P2	P3
.	.	.
.	**wait (mutex);**	.
wait (mutex);	**<S2>;**	.
<S1>;	**signal (mutex);**	**wait (mutex)**
signal (mutex);	.	**<S3>;**
.	.	**signal (mutex)**
.	.	.

- S1, S2, S3 sezioni critiche relative alla stessa risorsa
- mutex semaforo (binario) di mutua esclusione

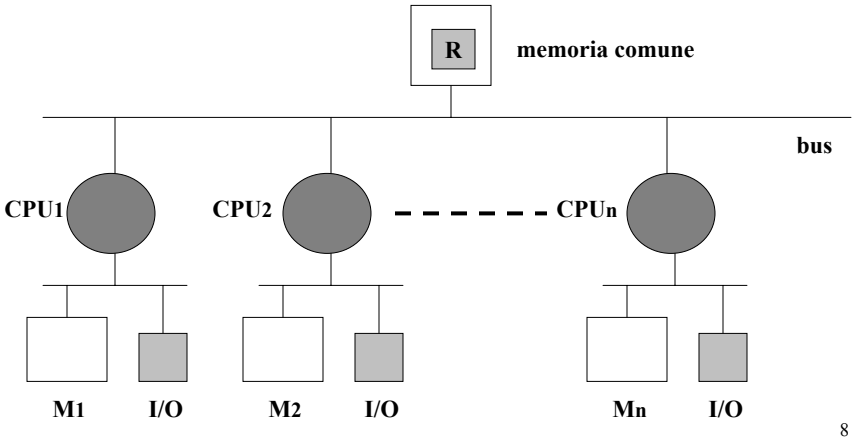
mutex0 = 1

Qualunque sia la sequenza di esecuzione dei processi la soluzione è sempre corretta

7

MUTUA ESCLUSIONE: ALCUNI PROBLEMI

1. E' sempre necessario usare **wait** e **signal** per assicurare la mutua esclusione (**overhead**)?
2. Come si ottiene la **non interrompibilità** nel caso di sistemi multiprocessori?



8

SOLUZIONE AL PRIMO PROBLEMA

Ipotesi: SEZIONI CRITICHE SUFFICIENTEMENTE BREVI

a) Sistema monoprocessore

P1	**P2**
.	.
<disabilita interruzioni>;	**<disabilita interruzioni>;**
<S1>;	**<S2>;**
<riabilita interruzioni>;	**<riabilita interruzioni>;**
.	.
.	.
.	.

9

b) Sistema multiprocessore

```
LOCK(x) {
   do while (x != 1)
      x = 0;
}

UNLOCK(x) {
   x = 1;
}
```

P1	**P2**
.	.
.	.
lock(x);	**lock(x);**
<S1>;	**<S2>;**
unlock(x);	**unlock(x);**
.	.

x = 1 risorsa libera

x = 0 risorsa occupata

10

- Problema dell'attesa attiva (**busy waiting**)
- Nell'ipotesi che l'hardware garantisca la mutua esclusione solo a livello di lettura o scrittura di una cella di memoria **solo unlock è indivisibile**
- Istruzione di **TEST AND SET LOCK (TSL)**
  - Copia il valore di **x** in un registro ed inserisce in **x** il valore **0**, in modo **indivisibile**
  - La CPU che esegue TSL tiene occupato il bus di memoria per impedire ad altre CPU di accedere alla memoria

11

lock:

```
tsl register, x     {copia x nel registro e pone x=0}
cmp register, 1     {x vale 1?}
jne lock            {se x=0 riinizia il ciclo}
ret                 {ritorna al chiamante; accesso alla sezione critica}
```

unlock:

```
mov x,1             {inserisce il valore 1 in x}
ret                 {ritorna al chiamante}
```

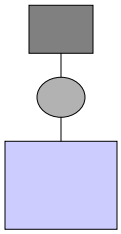
12

SOLUZIONE AL SECONDO PROBLEMA

Nel caso generale in cui **wait** e **signal** siano eseguite su processori diversi si ha:

```
wait(mutex) {
   lock(x);
   /*codice della wait*/
   unlock(x)
}
```

```
signal(mutex) {
   lock(x);
   /*codice della signal*/
   unlock(x)
}
```



x

mutex

R

13

COOPERAZIONE TRA PROCESSI CONCORRENTI

- Scambio di messaggi generati da un processo e consumati da un altro
- Scambio di un segnale temporale che indica il verificarsi di un dato evento

La *cooperazione tra processi* prevede che l'esecuzione di alcuni di essi risulti *condizionata* dall'informazione prodotta da altri

(vincoli sull'ordinamento nel tempo delle operazioni dei processi)

14

Esempio:

n processi P1, P2, …. Pn attivati ad intervalli prefissati di tempo da P0

- l'esecuzione di Pi non può iniziare prima che sia giunto il segnale da P0
- ad ogni segnale inviato da P0 deve corrispondere una attivazione di Pi

Processo Pi:

```
begin
  .
  .
  repeat
     wait (si);
     .
     .
  forever
end
```

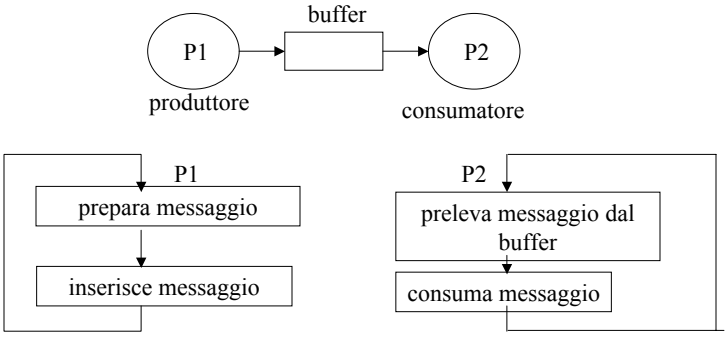
valore iniziale si0 = 0

Processo P0:

```
begin
  .
  .
  repeat
     .
     signal (si);
     .
     .
   forever
end
```

15

COMUNICAZIONE



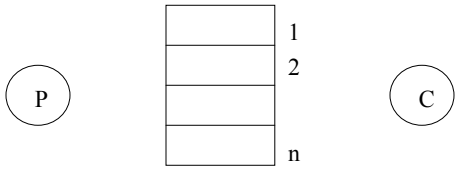
Sequenza corretta: Inserimento-prelievo-inserimento-prelievo….

Sequenze errate:

- Inserimento-inserimento-prelievo….
- Prelievo-prelievo-inserimento

16

Esempio: Produttore Consumatore



- Il produttore non può inserire un messaggio nel buffer se questo è pieno
- Il consumatore non può prelevare un messaggio dal buffer se questo è pieno

Se

d = numero dei messaggi depositati

e = numero dei messaggi estratti

N = numero dei messaggi che può contenere il buffer

0 <= d-e <= N

17

Processo produttore:

begin
  repeat
    <produzione messaggio>;
    <deposito messaggio>;
    signal (messaggio disponibile);
  forever
end

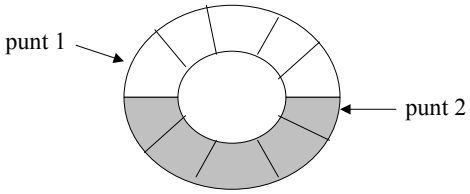
Processo consumatore:

begin
  repeat
    wait (messaggio disponibile)
    <prelievo di un messaggio>
    <consumo del messaggio>
  forever
end

Messaggio disponibile: valore iniziale = 0

18

Produttore e consumatore non devono mai accedere contemporaneamente alla stessa posizione del buffer



Inserimento:

buffer (punt 1) := messaggio prodotto

punt 1 := punt 1 + 1 (mod N)

Prelievo:

messaggio prelevato := buffer (punt 2)

punt 2 := punt 2 + 1 (mod N)

punt 1 = punt 2 significa:

a) buffer tutto pieno → solo il consumatore può agire sul buffer

b) buffer tutto vuoto → solo il produttore può agire sul buffer

19

Si introducono due semafori:

- **spazio disponibile** valore iniziale N
- **messaggio disponibile** valore iniziale 0

Processo produttore:

```
begin
  α: <produzione messaggio>;
     wait (spazio disponibile);

     <deposito messaggio>;

     signal (messaggio disponibile);
     go to α;
end
```

Processo consumatore:

```
begin
    α: wait (messaggio disponibile);

    <prelievo di un messaggio>,

     signal (spazio disponibile);
     <consumo del messaggio>;
     go to α;
end
```

20

Più produttori e più consumatori:

Processo produttore:

```
begin
   α: <produzione messaggio>;
      wait (spazio disponibile);
      wait (mutex1)
      <inserimento messaggio>;
      signal (mutex1)
      signal (messaggio disponibile);
      go to α;
end
```

Processo consumatore:

```
begin
   α: wait (messaggio disponibile);
      wait (mutex1)
      <prelievo messaggio>,
      signal (mutex1)
      signal (spazio disponibile);
      <consumo messaggio>;
      go to α;
end
```

mutex1 *valore iniziale = 1*

mutex2 *valore iniziale = 1*

21

ESEMPI DI USO DEI SEMAFORI

Si abbiano **n** unità di uno stesso tipo di risorsa R1, R2,…Rn (tutte equivalenti fra loro) ed **m** processi P1, P2, … Pm che devono operare su una qualunque di esse, Rj, mediante **certe operazioni** {A, B, …} in **modo esclusivo.**

I Soluzione

Ad ogni risorsa Rj viene assegnato un semaforo Mj con valore iniziale = 1 (semaforo di mutua esclusione sulla risorsa):

processo Pi:

```
……
wait (Mj)
Rj.A          Rj.A rappresenta l'operazione A eseguita su Mj
signal (Mj)
……
```

Inconvenienti della soluzione:

- Come decide il generico processo su quali risorse operare (come viene scelta j)?
- Può capitare che, una volta scelta Rj, se su di essa sta operando in quel momento un secondo processo Pk, il processo Pj si blocchi su wait(Mj), pur essendo disponibili altre risorse Rh (h<>j).

22

II Soluzione

Viene introdotta una nuova risorsa G, **gestore** di R1, R2, … Rn. Essa può essere concepita come **una struttura dati** destinata a mantenere lo stato delle risorse gestite. Sul gestore si opera tramite due procedure: **Richiesta** e **Rilascio**.

Procedure **Richiesta** (var x:1..n)
Procedure **Rilascio** (x:1..n)

*dove x rappresenta **l'indice** della risorsa **assegnata** o **rilasciata***

Struttura dati del gestore:

le procedure Richiesta e Rilascio dovranno essere eseguite in **mutua esclusione**

⇨ semaforo *mutex* di mutua esclusione con v.i. = 1

Un processo che esegue Richiesta chiede se vale la condizione "esiste una Rj disponibile". Un processo che esegue Rilascio segnala la validità della precedente condizione

⇨ semaforo RIS con valore iniziale = n

E' necessario un vettore di variabili booleane **Libero[i]** per registrare quale risorsa è in un certo istante libera (Libero[i] = true) e quale occupata (Libero[i] = false).

23

II Soluzione - segue

```
var   mutex : semaforo
      Ris : semaforo
      Libero : array [1..n] of boolean
{inizializzazione}
begin
     mutex := 1;
     Ris := n;
     for i := 1 to n do Libero[i] := true;
end
```

24

II Soluzione - segue

```
Procedure Richiesta (var x:1..n);
var i:  0..n;
begin
      wait(Ris);
      wait(mutex);
      i:=0;
      repeat
            i:= i + 1
      until Libero[i];
      x := 1;
      Libero[i] := false;
      signal(mutex);
end
```

```
Procedure Rilascio (var x:1..n);
var i:  0..n;
begin
      wait(mutex);
      i:=x;
      Libero[i] := true;
      signal(mutex);
      signal(Ris);
end
```

25

REALIZZAZIONE DI POLITICHE DI GESTIONE DELLE RISORSE

Nei problemi di sincronizzazione visti precedentemente si ha che:

- La decisione se un processo può proseguire l'esecuzione dipende dal valore di un solo semaforo ("mutex", "spazio disponibile", "messaggio disponibile")
- La scelta del processo da riattivare avviene tramite l'algoritmo implementato nella **signal** (FIFO).

In problemi di sincronizzazione **più complessi** si ha che:

- La decisione se un processo può proseguire l'esecuzione dipende in generale dal verificarsi di una **condizione di sincronizzazione**
- La scelta del processo da riattivare può avvenire sulla base di **priorità tra processi**

26

PROBLEMA DEI “READERS AND WRITERS”

- Processi lettori possono usare la risorsa contemporaneamente
- Processi scrittori hanno accesso esclusivo alla risorsa
- Processi lettori e scrittori si **escludono mutuamente** nell’uso della risorsa

I Soluzione

Un processo lettore aspetta solo se la risorsa è già stata assegnata ad un processo scrittore: cioè nessun lettore aspetta se uno scrittore è già in attesa (possibilità di attesa infinita da parte dei processi scrittori)

II Soluzione

Un processo lettore aspetta se un processo scrittore è in attesa (possibilità di attesa infinita da parte dei processi lettori)

27

I Soluzione

var readcount : **integer**(v.i.=0)

mutex, w : semaphore (v.i.=1)

READER	**WRITER**

wait(mutex);
readcount := readcount + 1;
if readcount = 1 **then** wait(w);
signal(mutex);
……
<lettura>
……..
wait(mutex)
readcount := readcount - 1;
if readcount = 0 **then** signal(w);
signal(mutex);
	wait(w);
	……
	<scrittura>
	……
	signal(w)

28

II Soluzione

var readcount, writecount : **integer**(v.i.=0)
mutex1, mutex2, mutex3, w, r : semaphore (v.i.=1)

READER	WRITER
wait(mutex3);	wait(mutex2);
wait(r);	writecount := writecount + 1;
wait(mutex1);	**if** readcount = 1 **then** wait(r);
readcount := readcount + 1;	signal(mutex2)
if readcount = 1 **then** wait(w);	wait(w);
signal(mutex1);	……
signal(r);	<scrittura>
signal(mutex3);	……
……	……
<lettura>	……
……..	
wait(mutex1)	signal(w)
readcount := readcount - 1;	wait(mutex2)
if readcount = 0 **then** signal(w);	writecount := writecount - 1;
signal(mutex1);	**if** writecount = 0 **then** signal(r);
	signal(mutex2);

29