ESERCIZI DI SINCRONIZZAZIONE TRA PROCESSI IN AMBIENTE GLOBALE

ESERCIZIO SincrProcAmbGlob-1

Si consideri un sistema nel quale è definito il semaforo *sem1* e i processi P1, P2 e P3.
Al tempo *t* il semaforo *sem1* ha la seguente configurazione:
Sem1: valore 0, coda P3
Allo stesso tempo, la CodaPronti contiene solo P2 e il processo P1 è in esecuzione.
Lo scheduler dei processi avviene con politica FIFO, senza prerilascio.
Si chiede come si modificano il semaforo *sem1* e la CodaPronti e quale processo è in esecuzione se si verificano (in alternativa) le due seguenti sequenze di eventi:

a) P1 esegue *wait (Sem1)* e successivamente il processo in esecuzione esegue *signal(Sem1);*
b) P1 esegue *signal (Sem1)* e successivamente il processo in esecuzione esegue *signal(Sem1);*

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti	Sem1
a-1	P1 esegue *wait (Sem1)*	P2	∅	0, P3→P1
a-2	Il processo in esecuzione esegue *signal(Sem1)*	P2	P3	0, P1
b-1	P1 esegue *signal (Sem1)*	P1	P2→P3	0, ∅
b-2	Il processo in esecuzione esegue *signal(Sem1)*	P1	P2→P3	1, ∅

ESERCIZIO SincrProcAmbGlob-2

Si consideri un sistema nel quale sono definiti il semaforo *sem* e i processi P1 (con priorità 1), P2 (con priorità 1) e P3 (con priorità 2). Lo scheduling avviene con una politica a priorità, che prevede il prerilascio e assegna il processore al processo pronto di priorità più elevata (a pari priorità applica la politica FIFO). La politica applicata al semaforo è la FIFO.
Al tempo *t* il semaforo *sem* ha valore 0, e la sua coda contiene P3.
Allo stesso tempo, il processo P1 è in esecuzione il processo P2 è pronto.
Come si modificano il semaforo *sem* e la CodaPronti e quale processo è in esecuzione se si verificano (in alternativa) le seguenti sequenze di eventi:

c) P1 esegue *wait (sem))* e successivamente il processo in esecuzione esegue *signal(sem);*
d) P1 esegue *signal (sem)* e successivamente il processo in esecuzione esegue *signal(sem);*

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti	Valore di *sem*	Coda di *sem*
a-1	P1 esegue *wait (sem)*	P2	∅	0	P3 → P1
a-2	Il processo in esecuzione esegue *signal(sem)*	P3	P2	0	P1
b-1	P1 esegue *signal (sem)*	P3	P2 → P1	0	∅
b-2	Il processo in esecuzione esegue *signal(sem)*	P3	P2 → P1	1	∅

ESERCIZIO SincrProcAmbGlob-3

Si consideri un sistema nel quale sono definiti il semaforo *sem* e i processi P1 (con priorità 1), P2 (con priorità 1) e P3 (con priorità 2). Lo scheduling avviene con una politica a priorità, che prevede il prerilascio e assegna il processore al processo pronto di priorità più elevata (a pari priorità applica la politica FIFO). La politica applicata al semaforo è la FIFO.
Al tempo *t* il processo P1 è in esecuzione il processo P2 è pronto; inoltre il semaforo *sem* ha valore 0 e la sua coda contiene P3. Dopo il tempo *t* si verifica la seguente sequenza di eventi:

e) P1 esegue *wait (sem))*
f) il processo in esecuzione esegue *signal(sem);*
g) il processo in esecuzione esegue *wait (sem)*
h) il processo in esecuzione esegue *signal(sem);*

Si chiede di specificare quale processo è in esecuzione dopo ogni evento e inoltre come si modificano il valore e la coda del semaforo *sem* e la *CodaPronti* .

SOLUZIONE

Sequenza di eventi	In Esecuzione	Coda Pronti	Valore di *sem*	Coda di *sem*
1) P1 esegue *wait (sem)*	P2	∅	0	P3 → P1
2) il processo in esecuzione esegue *signal(sem)*	P3	P2	0	P1
3) il processo in esecuzione esegue *wait (sem)*	P2	∅	0	P1 → P3
4) Il processo in esecuzione esegue *signal(sem)*	P2	P1	0	P3

ESERCIZIO SincrProcAmbGlob-4

In un sistema operativo con processi definiti in ambiente globale, i processi A (produttore) e B (consumatore) cooperano scambiandosi messaggi attraverso un buffer di 10 celle, ciascuna capace di contenere un messaggio e utilizzando i semafori *mutex* (valore iniziale 1), *CelleLibere* (valore iniziale 10) e *MessaggiGiacenti* (valore iniziale 0).
A meno delle operazioni di sincronizzazione e di quelle per la mutua esclusione (necessarie perché si supone che le operazioni *insert(item)* e *remove_item* non siano indivisibili), i frammenti rilevanti dei processi A e B sono i seguenti:

```
A (produttore):
........
While (true) {
      item = produce_item();
      insert(item);
      }
........

B (consumatore):
........
While (true) {
      item= remove_item();
      consume(item);
      }
........
```

Completare questi frammenti con le operazioni per la sincronizzazione e la mutua esclusione.

SOLUZIONE

```
A (produttore):
........
While (true) {
      item = produce_item();
      wait(&CelleLibere);
      wait(mutex);
      insert(item);
      signal(mutex);
      signal(&MessaggiGiacenti);
      }
........

B (consumatore):
........
While (true) {
      wait(&MessaggiGiacenti)
      wait(mutex);
      item= remove_item();
      signal(mutex);
      signal(&CelleLibere);
      consume(item);
      }
........
```

ESERCIZIO SincrProcAmbGlob-5 (Problema dei lettori e degli scrittori)

Una struttura dati (*base dati*) è condivisa da un insieme di processi, che appartengono a una delle due seguenti categorie:

1) *Processi lettori*: accedono alla *base dati* esclusivamente per leggere, senza modificare i dati;
2) *Processi scrittori*: accedono alla *base dati* senza restrizioni, con la possibilità di modificare i dati.

Per evitare interferenze, sono imposti seguenti vincoli:

- i processi scrittori accedono alla base dati in mutua esclusione, rispetto agli altri scrittori e ai lettori;
- i processi lettori accedono alla base dati in mutua esclusione rispetto agli scrittori, ma senza vincolo di mutua esclusione rispetto agli altri lettori (in altre parole, più lettori possono utilizzare la base dati concorrentemente).

Il problema viene risolto con una politica che consente l'accesso alternativamente a uno scrittore e a un insieme di lettori, ed evita l'attesa indefinita per gli scrittori. Precisamente, per le richieste valgono le seguenti clausole:

a) Quando la base dati non è utilizzata da nessun processo:
   a1) accede il primo processo (lettore o scrittore) che ne fa richiesta;
b) quando la base dati è utilizzata da uno scrittore, oppure da uno o più lettori:
   b1) se un lettore richiede l'accesso e la base dati è utilizzata da uno scrittore, il lettore richiedente viene sospeso;
   b2) se un lettore richiede l'accesso e la base dati è utilizzata da uno o più lettori, il lettore ottiene l'accesso *a condizione che non vi siano scrittori in attesa*;
   b3) se un lettore richiede l'accesso, la base dati è utilizzata da uno o più lettori e vi è almeno uno scrittore in attesa di accedere, il lettore richiedente viene sospeso *(questa clausola evita l'attesa indefinita per gli scrittori)*;
   b4) se uno scrittore richiede l'accesso e la base dati è utilizzata da uno scrittore oppure da uno o più lettori, lo scrittore si sospende.

Per i rilasci valgono le seguenti clausole:

c) quando uno scrittore rilascia la base dati:
   c1) se esistono lettori in attesa, tutti questi lettori sono riattivati e ottengono l'accesso;
   c2) se non esistono lettori in attesa ma esiste almeno uno scrittore in attesa, il primo di questi scrittori ottiene l'accesso;
   c3) se non esistono lettori o scrittori in attesa, si torna al caso a).
d) quando un lettore rilascia la base dati:
   d1) se altri lettori stanno ancora utilizzando la base dati, continua l'accesso in lettura da parte di questi processi;
   d2) se non vi sono altri lettori che utilizzano la base dati ed esiste almeno uno scrittore in attesa, il primo di questi scrittori ottiene l'accesso;
   d3) altrimenti si torna al caso a).

Per la soluzione del problema, si utilizzano i seguenti dati condivisi da tutti i processi:

- *LettoriAmmessi*: intero non negativo; valore iniziale 0
- *LettoriInAttesa*: intero non negativo; valore iniziale 0
- *ScrittoreAmmesso*: intero non negativo; valore iniziale 0
- *ScrittoriInAttesa*: intero non negativo; valore iniziale 0

e i seguenti semafori:

- *mutex* (valore iniziale 1): semaforo utilizzato per la mutua esclusione sulle sezioni critiche con le quali i processi verificano la condizione di accesso;
- *AttesaLettori (*valore iniziale 0*):* semaforo utilizzato per la sospensione dei lettori;
- *AttesaScrittori (*valore iniziale 0*):* semaforo utilizzato per la sospensione degli scrittori; valore iniziale 0.

Nota importante: per evitare lo stallo, la sospensione sui semafori *AttesaLettori* e *AttesaScrittori* deve avvenire dopo aver rilasciato la mutua esclusione instaurata per la verifica delle condizioni di accesso alla base dati. A questo scopo i processi registrano l'esito della verifica della condizione di accesso nella variabile locale *OK* e a seconda del valore assegnato a questa variabile, eseguiranno eventualmente la primitiva *wait* dopo aver rilasciato la mutua esclusione.

Si chiede di completare lo pseudo-codice sotto riportato, inserendo opportunamente le operazioni sui semafori *mutex, AttesaLettori* e *AttesaScrittori*.

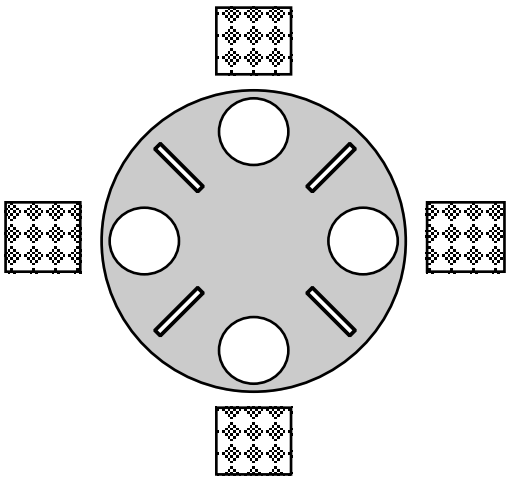
SOLUZIONE

```
ProcessoLettore_i
{
   ……..
   // prologo dell'accesso in lettura//
   OK = 0 ;
   wait(mutex);
   if ScrittoriInAttesa == 0 and ScrittoreAmmesso == 0   {
   // clausole a1, b2 e b3 //
        LettoriAmmessi ++; OK= 1;
        }
   else LettoriInAttesa ++;  / /clausola b1 //
   signal(mutex);
   if OK == 0  wait(AttesaLettori);
   // la sospensione avviene dopo il rilascio della mutua esclusione, per evitare lo stallo //
  < esegue accesso in lettura >
   // epilogo dell'accesso in lettura//
   wait(mutex);
   LettoriAmmessi -- ;
   if LettoriAmmessi== 0  and  ScrittoriInAttesa > 0        {
        ScrittoreAmmesso = 1; ScrittoriInAttesa --;
        // clausola d2 //
         signal(AttesaScrittori);
        }
   signal(mutex)
   ……..
}

ProcessoScrittore_j
    {
   …….
   // prologo dell'accesso in scrittura//
   OK = 0;
   wait(mutex);
   if LettoriAmmessi== 0 and ScrittoreAmmesso == 0     {
         ScrittoreAmmesso =1; OK= 1;  // clausola a1 //
        }
   else ScrittoriInAttesa ++;   // clausola b4 //
   signal(mutex);
   if OK == 0
        wait(AttesaScrittori);
  < esegue accesso in scrittura >
  // epilogo dell'accesso in scrittura//
   wait(mutex);
   ScrittoreAmmesso = 0;
   if LettoriInAttesa > 0   {
        while LettoriInAttesa> 0   {
        LettoriInAttesa --; LettoriAmmessi++;   // clausola c1 //
        signal(LettoriInAttesa);
        }
   else if ScrittoriInAttesa > 0     {
                ScrittoreAmmesso = 1; ScrittoriInAttesa --;   // clausola c2 //
                signal(AttesaScrittori);
                }
        }
   signal(mutex)
}
```

ESERCIZIO SincrProcAmbGlob-6 (Problema dei filosofi a cena)

N filosofi si ritrovano a cena in un ristorante cinese, occupando un tavolo circolare, apparecchiato con un piatto per ogni filosofo e un bastoncino interposto fra ogni coppia di piatti adiacenti (vedere la figura). Per mangiare, ogni filosofo deve avere a disposizione i due bastoncini che si trovano rispettivamente alla sua sinistra e alla sua destra, entrando così in competizione con i due filosofi che siedono ai suoi lati. Il filosofo che non riesca ad ottenere ambedue i bastoncino (perchè almeno uno è in possesso di un suo vicino) si sospende in attesa di ottenerli.



Il tavolo dei filosofi (N= 4)

Ogni filosofo alterna periodi in cui medita a periodi in cui vuole mangiare: quando decide di mangiare richiede i bastoncini ed eventualmente si sospende in attesa di ottenerli; dopo aver mangiato torna a pensare rilasciando i due bastoncini.

Si vuole una soluzione che eviti lo stallo, adottando la strategia di assegnare i bastoncini al filosofo richiedente solo se entrambi sono disponibili. In caso contrario, il filosofo richiedente si sospende senza entrare in possesso di nessun bastoncino, e sarà riattivato da uno dei filosofi che siedono ai suoi lati quando questo rilascerà i propri bastoncini. La riattivazione è subordinata alla condizione che il filosofo sospeso possa ora ottenere entrambi i bastoncini che gli sono necessari per mangiare.

In questa soluzione i filosofi condividono il vettore *stato[i]* (i= 0 .. N-1), dove lo stato del filosofo di indice i può assumere i valori *"ha fame", mangia", "pensa"*, con il seguente significato:

stato "ha fame" == > richiede i due bastoncini
stato "mangia" == > possiede i due bastoncini
transizione "mangia" → *"pensa" == > rilascia i due bastoncini*

Si utilizzano inoltre i seguenti semafori:

- *mutex,* con valore iniziale 1, per la mutua esclusione sul vettore *stato;*
- il vettore di semafori *filosofo[i]* (i= 0 .. N-1) dove il semaforo *filosofo[i]* ha valore iniziale 0 ed è utilizzato per la sospensione del filosofo di indice *i*.

NOTA: come nel problema dei lettori/scrittori, la sospensione sul semafori *filosofo[i]* deve avvenire dopo aver rilasciato la mutua instaurata per la verifica della condizione di acquisizione dei bastoncini. A questo scopo i processi registrano l'esito della verifica della condizione nella variabile locale *OK* ed eseguono la primitiva *wait* con la quale eventualmente si sospenderanno solo dopo aver rilasciato la mutua esclusione.

Si chiede di completare lo pseudo-codice sotto riportato, inserendo opportunamente le operazioni sui semafori.

SOLUZIONE

```
filosofo (i)
{
while true {
   // il filosofo di indice i decide di mangiare: protocollo per mangiare //
   OK= 0;
   wait(mutex);
   stato[i]= HaFame;
   if  stato[(i- 1) mod N]<> mangia and stato[(i+ 1) mod N]<> mangia                    {
         stato[i]= mangia; OK= 1  //può ottenere ambedue i bastoncini//
         }
   signal(mutex);
   if OK= 0
      wait(filosofo[i]);
   // se non può ottenere ambedue i bastoncini, il filosofo si sospende (fuori della mutua esclusione)
   senza prelevare il bastoncino eventualmente disponibile//
  < il filosofo di indice i mangia >
  //il filosofo di indice i  ha finito di mangiare: protocollo per pensare //
  wait(mutex);
   stato[i]=pensa;
   if stato[(i- 1) mod N]== HaFame and stato[(i- 2) mod N]<> mangia                    {
         stato[(i- 1) mod N]= mangia;
         signal(filosofo[i-1 mod n]));
         // il filosofo i riattiva il filosofo (i-1) mod N se, oltre al bastoncino i, può ottenere anche il
         // bastoncino (i-1) mod N
         }
   if stato[(i+ 1) mod N]== HaFame and stato[(i+ 2) mod N]<> mangia                    {
         stato[(i+ 1) mod N]= mangia;
         signal(filosofo[i+1 mod n]));
         // {il filosofo i riattiva il filosofo (i+1) mod N se, oltre al bastoncino  (i+1) mod N, può ottenere
         // anche il bastoncino (i+2) mod N
         }
   signal(mutex)
   }
```

NOTA
Si consideri in alternativa la seguente soluzione, che utilizza un semaforo per ogni bastoncino, organizzati nel vettore bastoncino[i]. Per ogni i (i= 0 .. N- 1) il valore iniziale del semaforo è 1. Il programma eseguito dal generico filosofo di indice i (1= 0, .. N-1) è il seguente:

```
filosofo[i]
{
while true {
   // il filosofo di indice i decide di mangiare: protocollo per mangiare //
   wait (bastoncino[i]);
   {il filosofo i si sospende se non può ottenere il bastoncino situato alla sua sinistra}
    wait(bastoncino[(i+ 1) mod N])
   {il filosofo i si sospende se non può ottenere il bastoncino situato alla sua destra}
< il filosofo di indice i mangia >
   //il filosofo di indice i  ha finito di mangiare: protocollo per pensare //
   signal(bastoncino[i]); signal(bastoncino[(i+ 1) mod N])
   {il filosofo rilascia i due bastoncini situati alla sua sinistra e alla sua destra}
   }
}
```

Questa soluzione non ha i requisiti desiderati, perché può determinare stallo. Lo stallo si verifica se tutti i filosofi decidono contemporaneamente di magiare e ciascuno esegue l'operazione *wait (bastoncino[i])*, con la quale chiede e ottiene il bastoncino situato alla propria sinistra: a questo punto nessuno dei filosofi può ottenere,il bastoncino situato alla propria destra e tutti si sospendono irreversibilmente.

ESERCIZIO SincrProcAmbGlob-7 (Problema del barbiere dormiglione)

Nel problema del "barbiere dormiglione" si considera un negozio gestito da un unico barbiere, con una poltrona per il servizio dei clienti e un numero illimitato di sedie per l'attesa. Il barbiere e i clienti sono processi ad ambiente globale e la poltrona è una risorsa, che può essere assegnata a un cliente per il taglio dei capelli, oppure utilizzata dal barbiere per dormire. All'apertura del negozio e quando non ci sono clienti in attesa di servizio, il barbiere occupa la poltrona per dormire, fino all'arrivo del primo cliente.
Quando entra nel negozio, il generico cliente ha il seguente comportamento:

- se il barbiere è addormentato, lo risveglia provocando il rilascio della poltrona, che occupa immediatamente;
- altrimenti (ciò avviene quando il barbiere è attivo per eseguire il taglio dei capelli ad un altro cliente) si blocca, accomodandosi su una delle sedie in attesa del suo turno;
- quando arriva il suo turno, è riattivato dal barbiere ed occupa la poltrona;
- dopo il taglio dei capelli, paga ed esce dal negozio.

Dopo che un cliente ha occupato la poltrona, il barbiere esegue il taglio dei capelli e al termine:

- se ci sono clienti in attesa del proprio turno, riattiva il primo;
- altrimenti occupa la poltrona e si addormenta.

Il problema viene risolto utilizzando i seguenti dati condivisi:

- *BarbiereAddormentato: booleano;*
- *ClientiInAttesa: intero; valore iniziale 0;*

e i seguenti semafori:

- *mutex: valore iniziale 1 (per la mutua esclusione);*
- *AttesaBarbiere: valore iniziale 0 (utilizzato dal barbiere per addormentarsi sulla poltrona. L'attesa su questo semaforo implica l'assegnazione della poltrona al barbiere);*
- *AttesaTurno: valore iniziale 0 (per i clienti che attendono il taglio. Le modalità di utilizzo garantiscono che il valore di questo semaforo sia sempre 0)*
- *TaglioCapelli: valore iniziale 0 (il cliente si sospende su questo semaforo all'inizio del taglio dei capelli e viene riattivato dal barbiere alla fine. L'attesa su questo semaforo implica l'assegnazione della poltrona al cliente).*

Per ogni cliente è inoltre definita la variabile privata *attende* utilizzata per evitare la sospensione all'interno della sezione critica nella quale viene decisa la sospensione. Allo stesso fine il barbiere utilizza la variabile privata *dorme*.
Notare che la risorsa poltrona non viene gestita esplicitamente, perché le assegnazioni e i rilasci sono implicite nelle operazioni sui semafori *AttesaBarbiere* e *TaglioCapelli*.
Si chiede di completare il programma del barbiere e il frammento di codice che controlla i clienti che entrano nel negozio, inserendo le opportune operazioni sui semafori.

SOLUZIONE

```
Barbiere
{
BarbiereAddormentato= true; wait(AttesaBarbiere);
// la riga precedente inizializza il negozio ed è eseguita prima della generazione dei  processi "cliente" //
while(true)          {
        <esegue il taglio dei capelli>
        signal(taglioCapelli);
        wait(mutex);
        if ClientiInAttesa> 0          {
                // riattiva un cliente in attesa del turno e libera la poltrona //
                ClientiInAttesa –;  dorme= false
                signal(AttesaTurno);
                }
        else { BarbiereAddormentato= true; dorme= true; }
        // occupa la poltrona per dormire, sospendendosi //
        signal(mutex);
        if dorme= true  wait(AttesaBarbiere);  // la sospensione avviene dopo il rilascio della mutua esclusione //
        }
}



Cliente   //Frammento di codice//
{
……
< entra nel negozio >
wait(mutex);
if BarbiereAddormentato== true {
        BarbiereAddormentato= false; attende= false;
        signal(AttesaBarbiere);
        // sveglia il barbiere che libera la poltrona //
        }
else    {
        // il barbiere è sveglio e sta servendo un cliente //
        ClientiInAttesa ++; attende= true;
        }
signal(mutex);
if attende == true
        wait(AttesaTurno); // la sospensione avviene dopo il rilascio della mutua esclusione //
        // Il cliente attende il suo turno. Sarà riattivato dal barbiere e troverà la poltrona libera //
wait(TaglioCapelli);
// siede sulla poltrona e attende il taglio dei capelli. Al termine…//
<paga ed esce dal negozio>
// Fine del frammento di codice //
 }
```