Esercizi di Concorrenza in Preparazione all'Esame

Corso di Sistemi Operativi

Ingegneria e Scienze informatiche, Università di Bologna

Prof. Stefano Ferretti

ESERCIZIO N. 1

Sincronizzazione con Message Passing

Completare il seguente frammento di codice relativo alla esecuzione di due processi, utilizzando i costrutti di Message Passing in modo che l'ordine delle istruzioni eseguite dai due processi sia il seguente:

- Istr_A1;
- Istr_B1;
- La computazione di Istr_A2 deve avvenire concorrentemente a Istr_B2
- A e B effettuano un rendez-vous
- La computazione di Istr_A3 deve avvenire concorrentemente a Istr_B3.
- Istr_A4
- Istr_B4.

Si considerino la Send NON Bloccante e la Receive Bloccante.

Process A {	Process B {
Istr_A1;	Istr_B1;
Istr_A2;	Istr_B2;
Istr_A3;	Istr_B3;
Istr_A4;	Istr_B4;
}	}

ESERCIZIO N. 2

MONITOR: Ponte a senso unico alternato (con limite di auto sul ponte)

Un ponte molto stretto consente l'accesso esclusivamente a senso unico alternato. Le automobili possono accedere in una sola direzione per volta, ovvero possono passare le auto che vanno dalla sponda nord alla sud oppure quelle che vanno dalla sponda sud alla nord. Non esistono priorità tra i processi auto. Un processo auto può attraversare il ponte solo se non vi sono sopra processi dell'altro tipo.
Inizialmente il ponte è vuoto, non ci sono auto in attesa e la direzione corrente è nord-sud. Il numero massimo di auto che possono stare contemporaneamente sul ponte è MAX. Correggere e/o completare le seguenti procedure che propongono la sincronizzazione dei processi coinvolti in questo scenario, utilizzando i costrutti dei monitor. Fornire una soluzione che eviti deadlock e starvation.

Le variabili condivise sono le seguenti
Coda_SN: variabile condition sulla quale le auto in direzione sud-nord si sospendono e si risvegliano.
Coda_NS: variabile condition sulla quale le auto in direzione nord-sud si sospendono e si risvegliano.
In_coda_SN: variabile intera che indica il numero delle auto in attesa di salire sul ponte, che si muovono in direzione sud-nord.
In_coda_NS: variabile intera che indica il numero delle auto in attesa di salire sul ponte, che si muovono in direzione nord-sud.
Dir: variabile che indica la direzione corrente di attraversamento del ponte, può assumere valore SN (da Sud a Nord) o NS (da Nord a Sud).
N_auto: variabile intera che indica il numero delle auto che sono sul ponte in un certo momento.

```
monitor Ponte_a_senso_unico_alternato {
  int in_coda_SN, in_coda_NS, n_auto;
  condition coda_SN, coda_NS;
  typedef direzione = {SN, NS};
  direzione dir;
```

A. **Completare la procedura entra_ponte_SN invocata dal processo Auto_SN**

```
void entra_ponte_SN() {

   boolean entrato;

   entrato=false;

   while (not(entrato)) {
       if ((                                                    )
       {

            coda_SN.wait();

        }
       else {

            N_auto++;

      }
   }
}
```

B. **Completare la procedura esci_ponte_SN invocata dal processo Auto_SN**

```
void esci_ponte_SN() {

     N_auto--;

     if (N_auto == 0) {



  }
}
```

C. **Completare e/o correggere la procedura entra_ponte_NS invocata dal processo Auto_NS**

```
void entra_ponte_NS() {

   boolean entrato;

   entrato=false;

   while (not(entrato)) {

        if (                                                                    )
        {

              coda_NS.signal();

         }
        else {


       }
   }
}
```

D. **Completare e/o correggere la procedura esci_ponte_NS invocata dal processo Auto_NS**

```
void esci_ponte_NS() {


     N_auto++;



     If (N_auto == 0) {


  }
}
```

E. **inizializzare le variabili necessaria/e alla sincronizzazione richiesta**

```
void inizializzazione() {

    N_auto =
    In_coda_SN =
    In_coda_NS =

  }

}
```

ESERCIZIO N. 3

Sincronizzazione di 3 Processi coi Semafori.
Il Processo A sveglia il processo B e il C poi si sospende in attesa che B e C abbiano terminato le operazioni sulle variabili condivise x, y e z. Una volta che B e C hanno terminato le operazioni svegliano A, B termina e C si ferma.. A questo punto A stampa i valori delle variabili x e y e sveglia C che stampa il valore di z..

Si correggano e/o completino i seguenti frammenti di codice, considerando mutex inizializzato a 1 e gli altri semafori inizializzati a 0.

Process A {	Process B {	Process C {
V(SemC);	P(semB);	P(semC)
	y= y*2	y = y * x;
P(SemA);		
P(mutex) Print(x)	z= x + y	V(SemC);
Print(y)		
V(semB)	V(SemA);	Print(z);
}	}	}

ESERCIZIO N. 4

Fornire una descrizione di “**sezione critica**”.

ESERCIZIO N. 5 **(6 punti)**

MONITOR: Pasticceria Fagioli
Nella pasticceria Fagioli il pasticcere prepara i pasticcini da esporre sul bancone e i clienti acquistano i pasticcini scegliendoli e prendendoli direttamente dal bancone.
Il bancone ha una capacità massima di N pasticcini.
Quando il bancone è vuoto, i clienti si mettono in attesa che il pasticcere riempia nuovamente il bancone, producendo N pasticcini.
Una volta riempito il bancone, il pasticcere risveglia tutti i processi clienti in attesa dei pasticcini e poi si blocca.
Il cliente che acquista l’ultimo pasticcino, sveglia il pasticcere, affinché questo riempia nuovamente il bancone.
Ogni cliente acquista un solo pasticcino per volta.
Una volta acquistato il pasticcino, il cliente esce dalla pasticceria.
Inizialmente il bancone è pieno.

Correggere e/o completare le seguenti procedure che propongono la sincronizzazione dei processi coinvolti in questo scenario, utilizzando i costrutti dei monitor.

Il monitor utilizza le seguenti variabili:
pasticcere: variabile condition che viene usata per gestire la sincronizzazione del processo pasticcere;
cliente: variabile condition che viene usata per gestire la sincronizzazione dei processi clienti della pasticceria;
pasticcini_sul_bancone: intero che indica il numero di pasticcini disponibili;
clienti_in_attesa: intero che indica il numero di clienti in attesa dei pasticcini.

```
monitor Pasticceria_fagioli {
  condition pasticcere, cliente;
  int pasticcini_sul_bancone, clienti_in_attesa;
```

A. **Completare la procedura prendi_pasticcino (invocata dal processo cliente)**

```
void prendi_pasticcino {


  if (pasticcini_sul_bancone==0)

  pasticcini_sul_bancone --;



  if (pasticcini_sul_bancone ==0)



}
```

B. **Completare la procedura prepara_pasticcini (invocata dal processo pasticcere)**

```
void prepara_pasticcini {

   pasticcini_sul_bancone = N;


   while (clienti_in_attesa > 0) {



  }

}
```

ESERCIZIO N. 6

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 = 1; z = 1.

Process A {	Process B {	Process C {
V(Sem1);	P(Sem1);	P(Sem3);
x = x + z;	P(Sem2);	z = x;
V(Sem2);	y = y - z;	V(Sem2);
P(Sem1);	V(Sem3);	P(Sem 3);
Print(x);	P(Sem2);	z= 2 *z;
V(Sem3);	Print(y);	Print(z);
}	V(Sem1); }	}

Con quale ordine i tre processi stampano le variabili?
Qual è il valore delle tre variabili che viene infine stampato?

ESERCIZIO N. 6

MONITOR: Bagno Unisex (con limite di persone nel bagno)
In una stazione di servizio lungo la A14 c'è un bagno solo che può essere condiviso da uomini e da donne, ma non contemporaneamente. Quando all'interno si trovano delle donne allora possono entrare soltanto altre donne e gli uomini devono attendere all'esterno. Quando all'interno si trovano degli uomini allora possono entrare soltanto altri uomini e le donne devono attendere all'esterno. Non esistono priorità tra i processi uomini e i processi donne. Inizialmente il bagno è vuoto, non ci sono uomini e donne in attesa. Il numero massimo di persone che possono stare contemporaneamente nel bagno è MAX. Correggere e/o completare le seguenti procedure che propongono la sincronizzazione dei processi coinvolti in questo scenario, utilizzando i costrutti dei monitor. Fornire una soluzione che eviti deadlock e starvation.

Le variabili condivise sono le seguenti
Uomini: variabile condition sulla quale i processi uomini si sospendono e si risvegliano.
Donne: variabile condition sulla quale i processi donne si sospendono e si risvegliano.
Uomini_in_attesa: variabile intera che indica il numero dei processi uomini attesa di entrare nel bagno.
Donne_in_attesa: variabile intera che indica il numero dei processi donne in attesa di entrare nel bagno.
Tipo: variabile che indica il tipo di persone attualmente in bagno, può assumere valore "donne" o "uomini".
Persone_in_bagno: variabile intera che indica il numero delle persone che sono in bagno in un certo momento.

```
monitor Bagno_unisex {
  int donne_in_attesa, uomini_in_attesa, persone_in_bagno;
  condition uomini, donne;
  typedef tipo_persone = {donne, uomini};
  tipo_persone tipo;
```

A. **Completare e/o correggere la procedura donna_entra_bagno invocata dal processo Donna**

```
void donna_entra_bagno() {

   boolean entrata;

   entrata=false;

   while (not(entrata)) {

        if (
                                                                                )
        {
            donne.wait();
        } else {

              persone_in_bagno++;


      }
   }
}
```

B. **Completare e/o correggere la procedura donna_esce_bagno invocata dal processo Donna**

```
void donna_esce_bagno() {

     persone_in_bagno--;

     if (persone_in_bagno == 0) {

     } else {


  }
}
```

C. **Completare e/o correggere la procedura uomo_entra_bagno invocata dal processo Uomo**

```
void uomo_entra_bagno(){

  boolean entrato;

  entrato=false;

  while (not(entrato)) {

     if (
                                                                    )
     {

            uomini.signal();

      else {


     }
   }
 }
```

D. **Completare e/o correggere la procedura uomo_esce_bagno invocata dal processo Uomo**

```
void uomo_esce_bagno() {

    persone_in_bagno++;

    If (persone_in_bagno == 0) {

    }
    else {


  }
}
```

E. **inizializzare le variabili necessaria/e alla sincronizzazione richiesta**

```
void inizializzazione() {
   persone_in_bagno =
   uomini_in_attesa =
   donne_in_attesa =

   }

}
```

ESERCIZIO N. 7

Sincronizzazione di 3 Processi coi Semafori.
Il Processo B sveglia il processo A e il processo C, poi si sospende in attesa che A e C abbiano terminato le operazioni sulle variabili condivise x, y e z. Una volta che i processi A e C hanno terminato le operazioni svegliano il processo B; il processo A termina e il processo C si ferma. A questo punto, il processo B stampa i valori delle variabili x e y e sveglia il processo C che stampa il valore di z
Si correggano e/o completino i seguenti frammenti di codice, considerando mutex inizializzato a 1 e gli altri semafori inizializzati a 0.

```
Process A {

  P(semA);

  y= y*2;

  z= x + y;

  V(SemB);

}
```

```
Process B {

  V(SemC);

  P(SemB);

  P(mutex);
  Print(x);

  Print(y);

  V(semA);
}
```

```
Process C {

  P(semC);

  y = y * x;

  V(SemC);

  Print(z);
}
```

ESERCIZIO N. 8

Sincronizzazione di 3 Processi coi Semafori.
Completare le seguenti porzioni di codice, utilizzando Semafori e variabili condivise, in modo che a video compaia stampata ripetutamente la stringa ABACAB

Process A {
  while (true) {
 	
  print(“A”);
 	
  }
}
	Process B {
	  while (true) {
	
	  print(“B”);
	
	  }
	}
		Process C {
		  while (true) {
		
		  print(“C”);
		
		  }
		}

ESERCIZIO N. 9

Fornire una descrizione di Semaforo.

ESERCIZIO N. 10

Sincronizzazione con Message Passing

Completare il seguente frammento di codice relativo all'esecuzione di due processi, utilizzando i costrutti di Message Passing in modo che l'ordine delle istruzioni eseguite dai due processi sia il seguente:

- Istr_A1;
- Istr_B1;
- La computazione di Istr_A2 e Istr_A3 deve avvenire concorrentemente a Istr_B2;
- Istr_B3;
- La computazione di Istr_A4 deve avvenire concorrentemente a Istr_B4.

Si considerino la Send NON Bloccante e la Receive Bloccante.

Process A {	Process B {
Istr_A1;	Istr_B1;
Istr_A2;	Istr_B2;
Istr_A3;	Istr_B3;
Istr_A4;	Istr_B4;
}	}

ESERCIZIO N. 11

Ad un ricevimento di nozze vi è un tavolo con le pietanze calde e uno col le pietanze fredde, rispettivamente M1 e M2 porzioni di buffet.
Un cameriere per ognuno dei due tavoli si occupa di servire gli N invitati (con N > M1 e N > M1). Gli invitati si mettono in fila ad un tavolo e attendono che il cameriere li serva, dopo di che si mettono in fila al secondo tavolo e attendono che il cameriere li serve. Quando hanno il piatto con entrambi i tipi di buffet allora ritornano al loro posto. Gli invitati sono serviti uno per volta
Quando non ci sono invitati il cameriere si sospende in attesa del loro arrivo. Il primo invitato che arriva al buffet risveglia il cameriere, il quale lo serve. Quando le porzioni sono finite allora i camerieri svegliano il cuoco che riempie i tavoli rispettivamente con M1 e M2 porzioni di buffet. Nel frattempo il cuoco resta sospeso.
Inizialmente i tavolo sono pieni di porzione e nessun invitato è in attesa ai tavoli.

Correggere e/o completare le seguenti procedure che propongono una soluzione a questo problema utilizzando i costrutti dei monitor.

Il monitor utilizza le seguenti variabili:
cameriere_buffet_caldo: variabile condition che viene usata per svegliare il cameriere che serve il tavolo con il buffet caldo;
cameriere_buffet_freddo: variabile condition che viene usata per svegliare il cameriere che serve il tavolo con il buffet freddo;
invitati_buffet_caldo: variabile condition che viene usata per svegliare gli invitati in attesa del buffet caldo;
invitati_buffet_freddo: variabile condition che viene usata per svegliare gli invitati in attesa del buffet freddo;
cuoco: variabile condition che viene usata per svegliare il cuoco;
buffet_caldo: intero che indica il numero di pietanze calde disponibili;
buffet_freddo: intero che indica il numero di pietanze fredde disponibili;
invitati_in_attesa_buffet_caldo: intero che indica il numero di invitati che sono sospesi in attesa del buffet caldo;
invitati_in_attesa_buffet_freddo: intero che indica il numero di invitati che sono sospesi in attesa del buffet freddo.

```
monitor Buffet_matrimonio {

condition cuoco, invitati_buffet_freddo, invitati_buffet_caldo,
cameriere_buffet_caldo, cameriere_buffet_freddo;

int buffet_caldo, buffet_freddo, invitati_in_attesa_buffet_caldo,
invitati_in_attesa_buffet_freddo;

invitati = N;
porzioni_calde = M1;
porzioni_fredde = M2;
```

A. **Inizializzare le variabili necessarie alla sincronizzazione richiesta**

```
void inizializzazione() {

  buffet_caldo =
  buffet_freddo =
  invitati_in_attesa_buffet_caldo =
  invitati_in_attesa_buffet_freddo =

}
```

B. **Completare e/o correggere la procedura prendi_porzione_calda (invocata dal processo invitato)**

```
void prendi_porzione_calda {

servito = false;

while (!servito) {

   if (invitati_in_attesa_buffet_caldo > 0) {

      invitati_in_attesa_buffet_freddo.wait ();

      cuoco.signal();

   } else {

      cameriere_buffet_freddo ++;

      cameriere_buffet_freddo.wait();

   }

 }
```

C. **Completare e/o correggere la procedura prendi_porzione_fredda (invocata dal processo invitato)**

```
void prendi_porzione_fredda {

servito = false;

while (!servito) {

   if (invitati_in_attesa_buffet_freddo > 0) {

       invitati_in_attesa_buffet_caldo.signal ();

       cameriere_buffet_caldo.signal();

   } else {

       cameriere_buffet_freddo --;

       cameriere_buffet_freddo.signal();

   }

 }
```

ESERCIZIO N. 12

Sincronizzazione di 3 Processi con i Semafori. Il Processo A sveglia prima il processo C e poi il processo B. Il processo A si sospende in attesa che B e C abbiano terminato le operazioni sulle variabili condivise x, y e z. Una volta che B e C hanno terminato le operazioni svegliano A. Il processo A opera sulle variabili condivise x e z, il processo C si sospende e il processo B stampa il valore della variabile y. A questo punto A stampa il valore della variabile x e sveglia il processo C che stampa il valore della variabile z. Si correggano e/o completino i seguenti frammenti di codice, considerando mutex inizializzato a 1 e gli altri semafori inizializzati a 0.

Process A {	Process B {	Process C {
V(SemA);	V(semB);	V(semC)
z= 2+x; x= x+z;	x = (x*y)+z;	y = z-(x*2);
Print(x);	P(SemA);	P(SemC);
Print(y);	}	Print(z); }
P(semB)		
}		

ESERCIZIO N. 13

Definire cosa si intende per starvation e descrivere un esempio in cui si verifica.

ESERCIZIO N. 14

Dato il seguente problema: gli spalatori di neve.

Il comune di Cesena manda N operatori a spalare la neve dal cortile dell'università.
Per poter spalare la neve gli operatori devono ottenere una pala, ma le pale in tutto sono M (M < N).
Gli operatori alternano momenti in cui spalano la neve a momenti in cui vanno al punto ristoro a prendere il caffè alla macchinetta per potersi scaldare.
Quando vanno a prendere il caffè, l'operatore rilascia la pala e un altro operatore può ottenerla per poter continuare a spalare.
Quando l'operatore ha finito di bere il caffè si mette in attesa di una pala libera per poter continuare a spalare.
Inizialmente tutte le pale sono libere e tutti gli operatori stanno bevendo il primo caffè della giornata.

Le variabili condivise sono le seguenti:

pale_libere: intero che indica il numero di pale non utilizzate dagli spalatori;
spalatori_in_attesa: intero che indica il numero degli spalatori in attesa di ottenere una pala per poter ricominciare a spalare;
spalatori: variabile condition, sulla quale si sospendono e si risvegliano gli spalatori;
stato[N]: indica lo stato in cui si trova lo spalatore e può assumere i seguenti valori "spala", "beve_caffè" e "aspetta_pala".

```
monitor spalatori_neve {

  int pale_libere, spalatori_in_attesa;

  condition spalatori;

  enum {spala,beve_caffè,aspetta_pala} stato[N];
```

A. **Completare e/o correggere la procedura prendi_pala invocata dallo spalatore)**

```
  void prendi_pala (int i) {

     stato[i] = aspetta_pala;

     verifica(i);

     if (stato[i] != spala) {

     }

}
```

B. **Completare e/o correggere la procedura verifica(i)**

```
void verifica (int i) {

   if (pale_libere>0) {

      stato[i] = spala;

   }
}
```

C. **Completare e/o correggere la procedura posa_pala**

```
void posa(int i) {

   stato[i] = beve_caffè;

}
```

ESERCIZIO N. 15

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 = 2; y = 1; z = 2.

```
Process A {
  P(Sem1);
  x = y + x;
  y = (x-z) + y;
  V(Sem3);
  P(Sem1);
  Print(y);
  x ++;
  V(Sem2);
}
```

```
Process B {
  V(Sem1);
  P(Sem2);
  y = x / y;
  z = z + y;
  V(Sem1);
  P(Sem2);
  y ++;
  Print(z);
  V(Sem3);
}
```

```
Process C {
  P(Sem3);
  z = (z+y) - x;
  x = x * y;
  V(Sem2);
  P(Sem3);
  Print(x);
  z --;
  Print(y);
  x = z + x;
}
```

Con quale ordine i tre processi stampano le variabili?

Qual è il valore delle tre variabili che viene infine stampato?

ESERCIZIO N. 16

Sincronizzazione di 3 Processi coi Semafori.
Completare le seguenti porzioni di codice, utilizzando Semafori e variabili condivise, in modo che a video compaia stampata ripetutamente la stringa CIAO

```
Process A {
  while (true) {

     print("C");

     print("A");

  }
}
```

```
Process B {
 while (true) {

     print("I");

  }
}
```

```
Process C {
 while (true) {

     print("O");

  }
}
```

ESERCIZIO N. 17

Sincronizzazione di 3 Processi coi Semafori.
Completare le seguenti porzioni di codice, utilizzando Semafori e variabili condivise, in modo che a video compaia stampata ripetutamente la stringa KISS

Process A {
  while (true) {
 	
  print(“K”);
 	
  }
}
	Process B {
	  while (true) {
	
	  print(“I”);
	
	  }
	}
		Process C {
		  while (true) {
		
		  print(“S”);
		
		  }
		}

ESERCIZIO N. 18

Sincronizzazione di 3 Processi con Message Passing.
Completare le seguenti porzioni di codice, utilizzando tecniche di Message Passing, in modo che a video compaia stampata ripetutamente la stringa KISS

Si considerino la Send NON Bloccante e la Receive Bloccante.

Process A {
  while (true) {
 	
  print(“K”);
 	
  }
}
	Process B {
	  while (true) {
	
	  print(“I”);
	
	  }
	}
		Process C {
		  while (true) {
		
		  print(“S”);
		
		  }
		}