Nome/cognome _____ N. di matricola (10 cifre) _____ Posizione: Riga _____ Col _____

UNIVERSITA' DI BOLOGNA - CORSO DI LAUREA IN INFORMATICA
CORSO DI SISTEMI OPERATIVI - ANNO ACCADEMICO 2003/2004
MIDTERM CONCORRENZA - 17 Novembre 2004

Esercizio -1: essersi iscritti correttamente per svolgere questa prova.
Esercizio 0: Su entrambi i fogli, scrivere correttamente nome, cognome, matricola e posizione prima di svolgere ogni altro esercizio.

Esercizio 1
Ci sono tanti produttori e un solo consumatore. I produttori e consumatori hanno la seguente vita:

```
process producer[i], i=1,..,n {
  prio=myprio();
  while(true) {
    x=produce();
    priopc.enqueue(x,prio);
  }
}
process consume {
  while(true) {
    y=priopc.dequeue;
    consume(y);
  }
}
```

La funzione myprio() restituisce un intero nel range 0,...,MAXPRIO. La coda ha capacità limitata MAX. Viene consumato per primo l'elemento con priorita' massima. Fra i processi con la stessa priorità l'ordine e' FIFO. Esiste poi un processo timer:

```
process timer {
  while(true) {
    sleep(Tick); // Dorme per Tick unità di tempo
    priopc.increaseage();
  }
}
```

increaseage incrementa la priorità a tutti i processi che non hanno priorità massima. Scrivere il monitor priopc.

Esercizio 2:
Un semaforo BiV e' un semaforo con due tipi diversi di V chiamati V1 e V2. L'invariante e':

nP <= min(nV1,nV2)+Init

dove nV1 e' il numero di V1 completate, nV2 e' il numero di V2 e nP il numero di P.
Questo semaforo e' equivalente ad un Semaforo ordinario?

Esercizio 3
Cosa stampa questo programma? Spiegare, facendo vedere (parte di) una delle possibili sequenze di passi.

```
j=2; i=3;
mutex = new Semaphore(1);
s1 = new Semaphore(1);
s2 = new Semaphore(2);
process P {
  while (j > 0) {
    s1.P()
    mutex.P();
    i++; j--;
    print A
    mutex.V();
    s2.V();
  }
}
process Q {
  while (i > 0) {
    s2.P();
    mutex.P();
    i--; j++;
    print B
    mutex.V();
    s1.V();
  }
}
```

Nome/cognome _____ N. di matricola (10 cifre) _____ Posizione: Riga _____ Col _____

UNIVERSITA' DI BOLOGNA - CORSO DI LAUREA IN INFORMATICA
CORSO DI SISTEMI OPERATIVI - ANNO ACCADEMICO 2003/2004
MIDTERM CONCORRENZA - 17 Novembre 2004

Esercizio -1: essersi iscritti correttamente per svolgere questa prova.
Esercizio 0: Su entrambi i fogli, scrivere correttamente nome, cognome, matricola e posizione prima di svolgere ogni altro esercizio.

Esercizio 1
Ci sono tanti produttori e un solo consumatore. I produttori e consumatori hanno la seguente vita:

```
process producer[i], i=1,..,n {
  prio=myprio();
  while(true) {
    x=produce();
    priopc.enqueue(x,prio);
  }
}
process consume {
  while(true) {
    y=priopc.dequeue;
    consume(y);
  }
}
```

La funzione myprio() restituisce un intero nel range 0,...,MAXPRIO. La coda ha capacità limitata MAX. Viene consumato per primo l'elemento con priorita' massima. Fra i processi con la stessa priorità l'ordine e' FIFO. Ogni N operazioni **dequeue**, tutti i produttori che non hanno priorità massima ricevono un incremento di 1 nella priorità.

Esercizio 2:
Un semaforo BiV e' un semaforo con due tipi diversi di V chiamati V1 e V2. L'invariante e':

nP <= max(nV1,nV2)+Init

dove nV1 e' il numero di V1 completate, nV2 e' il numero di V2 completate e nP il numero di P completate.
Questo semaforo e' equivalente ad un Semaforo ordinario?

Esercizio 3
Cosa stampa questo programma? Spiegare, facendo vedere (parte di) una delle possibili sequenze di passi.

```
j=3; i=5;
mutex = new Semaphore(1);
s1 = new Semaphore(2);
s2 = new Semaphore(3);

process P {
  while (j > 0) {
    s1.P()
    mutex.P();
    i++; j--;
    print A
    mutex.V();
    s2.V();
  }
}
process Q {
  while (i > 0) {
    s2.P();
    mutex.P();
    i--; j++;
    print B
    mutex.V();
    s1.V();
  }
}
```

Esercizio 2 (versione con max)

Da semafori biV a semafori ordinari:

```
Semaphore mutex;
Semaphore s;
int nV1;
int nV2;

biV(int init) {
  mutex = new Semaphore(1);
  s = new Semaphore(init);
}

P() {  s.P();  }

V1() {
  mutex.P();
  nV1++;
  if ( nV1 != nV2) {
    s.V();
  }
  mutex.V();
}
```

V2 simmetrico

Nota: dimensione degli int limitano nV1 e nV2.

Da semafori ordinari a semafori biV:

```
biV s;

Semaphore(int init) {  s = new biV(init);  }

P() {  s.P(); }

V() {
  s.V1();
}
```

Esercizio 2 (versione con min)

Da semafori biV a semafori ordinari:

```
Semaphore mutex;
Semaphore s;
int nV1;
int nV2;

biV(int init) {
  mutex = new Semaphore(1);
  s = new Semaphore(init);
}

P() {  s.P();  }

V1() {
  mutex.P();
  int prev = Min(nV1,  nV2);
  nV1++;
  if ( min(nV1, nV2) == prev+1) {
    s.V();
  }
  mutex.V();
}
```

V2 simmetrico

Nota: dimensione degli int limitano nV1 e nV2.

Da semafori ordinari a semafori biV:

```
biV s;

Semaphore(int init) {  s = new biV(init);  }

P() {  s.P(); }

V() {
  s.V1();
  s.V2();
}
```

Esercizio 3

Le due versioni sono simili. Considero quella con i semafori inizializzati a 1 e 2 per semplicità.
Viene stampata una stringa infinita, composta da un alternanza di A e B. Quali sono le regole di questa alternanza?

- Qualcuno ha scritto: stampa ABABAB etc continuamente alternato
  La risposta e' sbagliata, perché niente vincola i processi ad alternarsi in questo modo stretto. Per esempio, può stampare due B, poi due A, etc.
- Qualcuno ha scritto: stampa ABBBAAABBBAAA...
  Anche in questo caso, non e' detto che debba seguire un'alternanza così stretta
- Qualcuno ha scritto: stampa al massima due B consecutive e poi un A
  Ci stiamo avvicinando, ma ancora non va bene: può anche stampare più di due B e più di una A,
- Qualcuno ha scritto stampa al massimo tre A e al massimo tre B
  Ancora meglio: questa risposta si basa sull'osservazione che ci sono tre "risorse" (fra l'inizializzazione di S1 e l'inizializzazione di S2), e che A e B se le "passano" uno con l'altro. Ma scritto così, sembra che questa sequenza: ABBBABBBABBBABBB.... sia possibile, e non è vero.

La risposta corretta è:
sia p una qualunque prefisso della stringa infinita stampata
sia nA il numero di A comprese in p
sia nB il numero di B comprese in p

allora: $nB-2 <= nA <= nB+1$

Esercizio 1
Questa è una delle possibili soluzioni. Si considera anche la priorità al momento dell'inserimento, e mostra come sia possible gestire qualunque tipo di meccanismo di priorità.

due code:
Queue queue
Queue waiting

un contatore:
int size

```
p.e. void enqueue(Object x, int prio)
{
  if (size == MAX) {
    condition c = new condition();
    waiting[prio].add(c);
    c.wait();
  }
  size++;
  queue[prio].add(x);
  get.signal();
}

p.e. Object dequeue()
{
  if (size == 0)
    get.wait();
  Object obj = null;
  for (int i=MAXPRIO; i >=0 && obj != null; i--) {
    if (queue[i].size() > 0) {
      obj = queue[i].remove();
    }
  }
  size--;
  condition c = null;
  for (int i=MAXPRIO; i >=0 && c != null; i--) {
    if (waiting[i].size() > 0) {
      c = waiting[i].remove();
    }
  }
  if (c != null)
    c.signal();
  return obj;
}

p.e. increaseAge()
{
  queue[MAXPRIO].add(queue[MAXPRIO-1]);
  waiting[MAXPRIO].add(waiting[MAXPRIO-1]);
  for (int i=MAXPRIO-1; i >= 0; i++) {
    queue[i] = queue[i-1];
    waiting[i] = waiting[i-1];
  }
}
```