

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA DE TELECOMUNICACIÓ
DEPARTAMENT D'ENGINYERIA TELEMÀTICA
PROGRAMACIÓ CONCURRENT

Examen final. Tardor 03/04. Codi 230-11520-05-0

Instruccions:

- Omplir les dades personals, DNI i codi en el revers de l'enunciat
- Omplir tot el rectangle de la resposta correcta amb llapis segons la numeració de l'esquerra

NOM, DNI I GRUP: *Escriviu en el recuadre les vostres dades personals*

NOTES:

- *S'ha d'entregar el full de respostes i els fulls d'enunciat*
- *Marqueu amb un cercle la resposta correcta*
- *Suposeu que els identificadors referenciats en els fragments de codi estan correctament definits*
- *Per simplicitat, s'ha omès el codi de tractament d'excepcions*
- *Les respostes a les preguntes d'un mateix problema han de ser compatibles entre si*

PROBLEMA:

One lane bridge: Per un pont poden circular cotxes, modelats com a threads, en un o altre sentit, però no en ambdós sentits a la vegada. Es programa una solució amb pas de missatges pura amb clients remots, proxies o agents en l'extrem servidor i monitor actiu que fa el control d'accés. Els clients realitzen l'operació `enter(id, turn)` per sol·licitar l'entrada del cotxe `id` al pont en sentit `turn` i `exit()` per sortir del pont.

El codi del monitor actiu usa una variable `l` de tipus `ArrayList`, operacions `read_str`, `read_int` i `write_char` per llegir un `String`, un `int` i escriure un `char` a través d'un `Socket` respectivament i respon al següent esquema:

```
while (true) {
    if (cond) {
        A
    } else {
        B
    }
    if (str.equals("enter")) {
        C
    } else {
        D
    }
}
```

12. Dos productors, amb comportament idèntic, emeten en seqüència els números 1 i 2 amb el codi no atòmic `...println(...); buffer.put(i); i acaben`. Dos consumidors llegeixen dos dels números cadascun amb el codi no atòmic `...println(...+buffer.get()); i acaben`. Put i get són operacions d'un monitor buffer. Quin entrellaçat(s) és(són) incorrecte(s)?:

a) Consumidor(2) obté 1
Consumidor(1) obté 2
Consumidor(1) obté 1
Consumidor(2) obté 2 (1)

b) Consumidor(1) obté 1
Consumidor(2) obté 1
Consumidor(1) obté 2
Consumidor(2) obté 2 (3)

c) Consumidor(2) obté 2
Consumidor(1) obté 1
Consumidor(2) obté 1
Consumidor(1) obté 2 (2)

d) Cap dels entrellaçats (1), (2) i (3) és incorrecte

13. Per resoldre el problema N productors/M consumidors amb buffer de capacitat 1 calen com a mínim:

- a) 1 semàfor
- b) 2 semàfors
- c) 3 semàfors
- d) Cap de les altres respostes és correcta

14. El codi if (cond) {A} del problema **One lane bridge** és:

```
a) if (l.isEmpty()) {
    str = data.read_str();
    id = data.read_int();
    turn = data.read_int();
}

b) if (l.isEmpty()) {
    str = data.read_str();
    if (str.equals("enter")) {
        id = data.read_int();
        turn = data.read_int();
    }
}

c) if (l.isEmpty() || ncars == 0) {
    str = data.read_str();
    id = data.read_int();
    turn = data.read_int();
}

d) if (l.isEmpty() || ncars > 0) {
    str = data.read_str();
    if (str.equals("enter")) {
        id = data.read_int();
        turn = data.read_int();
    }
}
```

15. El codi B del problema **One lane bridge** és:

```
a) command c = (command) l.get();
str = c.str;

b) command c = (command) l.get();
str = c.str; id = c.id; turn = c.turn;

c) command c = (command) l.remove(0);
str = c.str;

d) command c = (command) l.remove(0);
str = c.str; id = c.id; turn = c.turn;
```

16. El codi C del problema **One lane bridge** és:

```
a) while (ncars > 0) && actual_turn != turn)
    l.add(new command(str, id, turn));
ncars++; actual_turn = turn;
ctl[id].write_char('a');

b) while (ncars > 0) && actual_turn != turn)
    l.add(new command(str, id, turn));
ncars++;

c) if (ncars > 0) && actual_turn != turn)
    l.add(new command(str, id, turn));
else {
    ncars++; actual_turn = turn;
    ctl[id].write_char('a');
}

d) if (ncars > 0) && actual_turn != turn)
    l.add(new command(str, id, turn));
ncars++; actual_turn = turn;
ctl[id].write_char('a');
```

17. El codi D del problema **One lane bridge** és:

```
a) if (actual_turn == turn) ncars--;
b) if (actual_turn == turn) ncars++;
c) ncars--;
d) if (actual_turn != turn) ncars--;
```

PROBLEMA:

ProductorsABCs: Una aplicació disposa de tres tipus de processos. Els processos **A** produeixen **a**'s, els processos **B** produeixen **b**'s i els processos **C** necessiten (consumeixen) una **a** i una **b** per a produir **ab**'s.

7. En el problema **ProductorsABCs** els processos d'un mateix tipus produeixen en exclusió mútua. Per exemple, dos processos **A** no poden estar produint alhora, però si que poden estar produint alhora un procés **A** i un **B** (o **C**). A més, els processos **A** i **B** no poden fer una nova producció fins que un procés **C** no ha consumit l'anterior. Quina resposta modela bé aquest comportament?

Els semàfors SA i SB estan inicialitzats a 1, i els SC, SCA, SCB a 0.

a)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SC.P();
//produir a	//produir a	SC.P();
SC.V();	SC.V();	//produir ab
SA.V();	SB.V();	

b)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SCA.P();
//produir a	//produir a	SCB.P();
SCA.V();	SCB.V();	//produir ab
SA.V();	SB.V();	

c)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SCA.P();
//produir a	//produir a	SCB.P();
SCA.V();	SCB.V();	//produir ab
SA.V();	SB.V();	SCA.V();
SCA.P();	SCB.P();	SCB.V();

d)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SCA.P();
//produir a	//produir a	SCB.P();
SCA.V();	SCB.V();	//produir ab
		SA.V();
		SB.V();

8. En el problema **ProductorsABCs** els processos d'un mateix tipus produeixen en exclusió mútua. Per exemple, dos processos **A** no poden estar produint alhora, però si que poden estar produint alhora un procés **A** i un **B** (o **C**). Els processos **A** i **B** poden fer una nova producció sense que un procés **C** hagi consumit l'anterior. Quina resposta modela bé aquest comportament?

Els semàfors SA i SB estan inicialitzats a 1, i els SC, SCA, SCB a 0.

a)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SC.P();
//produir a	//produir a	SC.P();

SC.V();	SC.V();	//produir ab
SA.V();	SB.V();	

b)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SCA.P();
//produir a	//produir a	SCB.P();
SCA.V();	SCB.V();	//produir ab
SA.V();	SB.V();	

c)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SCA.P();
//produir a	//produir a	SCB.P();
SCA.V();	SCB.V();	//produir ab
SA.V();	SB.V();	SCA.V();
SCA.P();	SCB.P();	SCB.V();

d)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SCA.P();
//produir a	//produir a	SCB.P();
SCA.V();	SCB.V();	//produir ab
		SA.V();
		SB.V();

9. En el problema **ProductorsABCs** els processos no produeixen en exclusió mútua. Per exemple, dos processos **A** poden estar produint alhora, i també poden estar produint alhora un procés **A** i un **B** (o **C**). Els processos **A** i **B** poden fer una nova producció sense que un procés **C** hagi consumit l'anterior. Quina resposta modela bé aquest comportament?

Els semàfors SA i SB estan inicialitzats a 1, i els SC, SCA, SCB a 0.

a)

Procés A	Procés B	Procés C
//produir a	//produir a	SC.P();
SC.V();	SC.V();	SC.P();
		//produir ab

b)

Procés A	Procés B	Procés C
//produir a	//produir a	//produir ab

c)

Procés A	Procés B	Procés C
SA.P();	SB.P();	SCA.P();
//produir a	//produir a	SCB.P();
SCA.V();	SCB.V();	//produir ab
SA.V();	SB.V();	

d)

Procés A	Procés B	Procés C
//produir a	//produir a	SCA.P();
SCA.V();	SCB.V();	SCB.P();
		//produir ab

10. Ara es vol resoldre el problema **ProductorsABCs**, sense exclusió mútua i amb possibilitat de més d'una producció pendent, utilitzant monitors. El codi dels processos és:

Procés A //produir a monitor.pa();	Procés B //produir a monitor.pb();	Procés C monitor.pab(); //produir ab
-------------------------------------------------	-------------------------------------------------	---------------------------------------------------

Quin codi per al monitor és correcte?

a)

```
private int na=0;
private int nb=0;
public synchronized void pa(){
    na++;
}
public synchronized void pb(){
    nb++;
}
public synchronized void pab(){
    while(na==0 || nb==0) wait();
    na--;
    nb--;
}
}
```

b)

```
private int na=0;
private int nb=0;
public synchronized void pa(){
    na++;
    notifyAll();
}
public synchronized void pb(){
    nb++;
    notifyAll();
}
public synchronized void pab(){
    if(na==0 || nb==0) wait();
    na--;
    nb--;
    notifyAll();
}
}
```

c)

```
private int na=0;
private int nb=0;
public synchronized void pa(){
    na++;
    notify();
}
public synchronized void pb(){
    nb++;
    notify();
}
public synchronized void pab(){
    if(na==0 || nb==0) wait();
    na--;
    nb--;
}
}
```

d)

```
private int na=0;
private int nb=0;
public synchronized void pa(){
```

```
na++;
notify();
}
public synchronized void pb(){
    nb++;
    notify();
}
public synchronized void pab(){
    while(na==0 || nb==0) wait();
    na--;
    nb--;
}
}
```

11. Hi ha N processos que realitzen una mateixa tasca de forma iterativa, però cap procés pot passar a la següent iteració, fins que tots els processos han acabat amb la iteració actual. El codi dels processos és

```
Procés
While(true){
    //Fer tasca
    monitor.esperar_resta();
}
}
```

Donats els següents codis del monitor. Quina és la resposta correcta?

a)

```
private nf=0;
public synchronized void esperar_resta(){
    nf++;
    while(nf<N) wait();
    nf--;
    notifyAll();
}
}
```

b)

```
private nf=0;
public synchronized void esperar_resta(){
    nf++;
    while(nf<N) wait();
    notifyAll();
    nf--;
    while(nf>0) wait();
}
}
```

c)

```
private nf=0;
public synchronized void esperar_resta(){
    nf++;
    while(nf<N) wait();
    notifyAll();
    nf--;
    while(nf>0) wait();
    notifyAll();
}
}
```

d) Cap de les anteriors.

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA DE TELECOMUNICACIÓ
DEPARTAMENT D'ENGINYERIA TELEMÀTICA
PROGRAMACIÓ CONCURRENT

Examen final. Tardor 03. Codi 230-11520-05-0

NOM, DNI I GRUP: *Escriviu en el recuadre les vostres dades personals*

PROBLEMA:

Se pretende implementar mediante Semáforos los **Monitores de Hoare**, es decir, con política **Signal and Urgent Wait**.

Política **Signal and Urgent Wait**: cuando un proceso señala (cond_signal()) sobre una condición en la que otro proceso está esperando, el proceso señalador debe dejar entrar en monitor al proceso señalado y éste salir de monitor automáticamente para esperarse en una cola prioritaria “urgent”. Por otro lado, cualquier proceso, antes de liberar la exclusión mútua, debe testear si hay algún otro proceso esperando en la cola prioritaria “urgent” para dejar entrar a uno de estos procesos de nuevo en monitor de forma prioritaria respecto a los que esperan en la cola de monitor.

Se definen las clases Monitor_Hoare y Condition_Hoare como se detalla a continuación. Se introduce en la definición de monitor un semaforo mutex para conseguir la exclusión mútua y un semáforo urgent para detener a los procesos que señalan.

```
public class Monitor_Hoare {  
  
    private Semaforo mutex, urgent;  
    private int urgentcount;  
  
    public Monitor_Hoare() {  
        mutex = new Semaforo(1);  
        urgent = new Semaforo(0);  
        urgentcount = 0; }  
  
    public void mon_enter() { ... }
```

```
    public void mon_exit() { ... }  
  
    public Condition_Hoare create_cond() {  
        return new Condition_Hoare(mutex,urgent,urgentcount); }  
}  
  
public class Condition_Hoare{  
  
    private Semaforo cond, mutex, urgent;  
    private int condcount, urgentcount;  
  
    public Condition_Hoare(Semaforo mutex, Semaforo urgent, int urgentcount)  
    {  
        this.mutex = mutex;  
        this.urgent = urgent;  
        this.urgentcount = urgentcount;  
        cond = new Semaforo(0);  
        condcount = 0; }  
  
    public void cond_wait() { ... }  
  
    public void cond_signal() { ... }  
}
```

12. ¿Cuál de estas afirmaciones es correcta?

- a) Los semáforos permiten una programación más estructurada que los monitores
- b) El paso de mensajes está más indicado para sistemas de memoria compartida
- c) Los semáforos están más indicados para sistemas de memoria distribuida
- d) Es posible simular el comportamiento de un semáforo general usando sólo semáforos binarios

13. En Java, siempre que se llama desde un método sincronizado de un objeto a un método no sincronizado de otro objeto distinto:

- a) Se mantiene la exclusión mutua en el acceso al primer objeto y al segundo. (1)
- b) Se mantiene la exclusión mutua en el acceso al primer objeto y no se mantiene en el acceso al segundo.

- c) (1) es correcta, pero al hacer un wait() en el segundo método, se liberan ambas exclusiones mutuas.
- d) Ninguna de las otras respuestas es correcta.
14. Una de estas afirmaciones es cierta en Java:
- a) Por defecto se proporciona exclusión mutua entre todos los métodos públicos de un objeto
- b) Sólo se proporciona exclusión mutua sobre los métodos privados de un objeto
- c) Un método sincronizado de un objeto se ejecuta en exclusión mutua con los métodos sincronizados de otros objetos
- d) Ninguna de las otras respuestas es correcta.

15. El código del método mon_entr() del problema de **Monitores de Hoare** es:

- a) `if(urgentcount>0) {urgent.P();}
else {mutex.P();}`
- b) `mutex.P();`
- c) `if(urgentcount>0) {urgent.P();}
mutex.P();`
- d) `urgent.P();`

16. El código del método mon_exit() del problema de **Monitores de Hoare** es:

- a) `mutex.V();`
- b) `if(urgentcount>0) {urgent.V();}
mutex.V();`
- c) `if(urgentcount>0) {mutex.V();}
else {urgent.V();}`
- d) `if(urgentcount>0) {urgent.V();}
else {mutex.V();}`

17. El código del método cond_wait() del problema de **Monitores de Hoare** es:

- a) `condcount++;
if(urgentcount>0) {urgent.V();}
else {mutex.V();}
cond.P();`

`condcount--;`

- b) `condcount++;
mutex.V();
cond.P();
mutex.P();
condcount--;`

- c) `condcount++;
if(urgentcount>0) {urgent.V();}
mutex.V();
cond.P();
condcount--;`

- d) `condcount++;
if(urgentcount>0) {urgent.V();}
mutex.V();
cond.P();
mutex.P();
condcount--;`

18. El código del método cond_signal() del problema de **Monitores de Hoare** es:

- a) `urgentcount++;
if(condcount>0) {cond.V();}
urgentcount--;`
- b) `urgentcount++;
if(condcount>0) {cond.V(); urgent.P();}
urgentcount--;`
- c) `urgentcount++;
cond.V();
urgent.P();
urgentcount--;`
- d) `urgentcount++;
cond.V();
mutex.V();
urgent.P();
urgentcount--;`

19) Les variables de condició:

- a) Es correcte declarar-les en qualsevol part del programa.
- b) Una mateixa variable de condició pot ser compartida per varis monitors.
- c) La suma de dues variables de condició, és una variable que inclou a tots els processos que s'esperaven en les variables que se sumen.
- d) Totes les altres respostes són falses.

Suposem la següent solució al problema dels lectors-escriptors.

```
int num_lec = 0, num_esc = 0;
Semaphore mutex_1 = new Semaphore(1);
Semaphore mutex_2 = new Semaphore(1);
Semaphore mutex_3 = new Semaphore(1);
Semaphore llegir = new Semaphore(1);
Semaphore escriure = new Semaphore(1);
```

PROCESSOS LECTORS:

```
mutex_3.P();
    llegir.P();
    mutex_1.P();
        num_lec = num_lec + 1;
        if (num_lec == 1) escriure.P();
    mutex_1.V();
    llegir.V();
mutex_3.V();
// llegir la base de dades
mutex_1.P();
    num_lec = num_lec - 1;
    if (num_lec == 0) escriure.V();
mutex_1.V();
```

PROCESSOS ESCRIPTOROS:

```
mutex_2.P();
    num_esc = num_esc + 1;
    if (num_esc == 1) llegir.P();
mutex_2.V();
escriure.P();
// modificar la base de dades
escriure.V();
mutex_2.P();
    num_esc = num_esc - 1;
    if (num_esc == 0) llegir.V();
mutex_2.V();
```

20) En aquesta solució, qui té preferència d'accés?

- a) Els processos lectors.
- b) Els processos escriptors.
- c) L'accés és alternatiu entre lectors i escriptors.
- d) L'accés és aleatori.

21) Si hi ha un escriptor actiu:

- a) Els processos lectors que intenten accedir a la base de dades s'aturen tots en el semàfor llegir.
- b) Els processos lectors que intenten accedir a la base de dades s'aturen tots en el semàfor mutex_3.
- c) Els processos lectors que intenten accedir a la base de dades s'aturen tots en el semàfor mutex_1.
- d) Cap de les altres respostes és certa.

Suposem que el següent monitor s'utilitza per controlar l'accés de varis processos a un recurs compartit.

```
class ControlAcces extends Monitor {
    protected boolean lliure = true;
    protected Condition torn;

    // Aquirir recurs
    public void adquirir(){
        mon_enter();
        if (!lliure) { torn.wait(); }
        lliure = false;
        mon_exit();
    }

    // Alliberar recurs
    public void alliberar(){
        mon_enter();
        lliure = true;
        torn.signal();
        mon_exit();
    }
}
```

22) Aquest monitor

- a) Funciona correctament si la disciplina és Signal and Continue.
- b) Funciona correctament si la disciplina és Signal and Urgent Wait.
- c) Funciona correctament amb les disciplines Signal and Continue i Signal and Urgent Wait.
- d) Funciona correctament amb les disciplines Signal and Continue i Signal and Urgent Wait, només si es canvia el signal per un signalAll.

Es vol simular un mecanisme de pas de missatges assíncron en JAVA. A tal efecte es construeix la següent classe

Canal. El mètode rebre és bloquejant en el cas en que no existeixin missatges en el canal.

```
public class Canal {
    private int num_miss;
    private Vector missatges;

    public Canal(){
        num_miss = 0;
        missatges = new Vector();
    }

    ... enviar (Object o) {
        ...
    }

    ... rebre() {
        ...
    }
}
```

Es demana implementar el codi dels mètodes enviar i rebre, de manera que les respostes siguin compatibles entre si.

23) El codi del mètode enviar és:

```
a) public synchronized void enviar(Object o) {
    num_miss--;
    missatges.addElement(o);
    if (num_miss > 0) notify();
}

b) public synchronized void enviar(Object o) {
    num_miss++;
    missatges.addElement(o);
    if (num_miss <= 0) notify();
}

c) public synchronized void enviar(Object o) {
    num_miss++;
    missatges.addElement(o);
    if (num_miss > 0) notify();
}

d) public synchronized void enviar(Object o) {
    num_miss--;
    missatges.addElement(o);
    notify();
}
```

24) El codi del mètode rebre és:

```
a)
public synchronized Object rebre() {
    Object missatge_rebut = null;
    num_miss++;
    missatge_rebut = missatges.remove(0);
    if(num_miss <0){
        try{
            wait();
        } catch (Exception e) {}
    }
    return missatge_rebut;
}

b)
public synchronized Object rebre() {
    Object missatge_rebut = null;
    num_miss--;
    if(num_miss <0){
        try{
            wait();
        } catch (Exception e) {}
    }
    missatge_rebut = missatges.remove(0);
    return missatge_rebut;
}

c)
public synchronized Object rebre() {
    Object missatge_rebut = null;
    try{
        wait();
    } catch (Exception e) {}
    num_miss--;
    missatge_rebut = missatges.remove(0);
    return missatge_rebut;
}

d)
public synchronized Object rebre() {
    Object missatge_rebut = null;
    num_miss++;
    if(num_miss <=0){
        try{
            wait();
        } catch (Exception e) {}
    }
    missatge_rebut = missatges.remove(0);
    return missatge_rebut;
}
```