Departamento de Informática

Convocatoria extraordinaria 2, Curso 2012-2013
Sistemas Concurrentes y Distribuidos

Nombre: DNI:

Cumplimentar el cuestionario en la tabla anexa. (**30 % nota global**).
+0.15 respuesta correcta, -0.05 por error, 0 por repuesta no contestada.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b	c	b	d	c	b	a	b	b	d	a	b	c	a	b

1) Para que un programa concurrente sea correcto, deben cumplirse las siguientes propiedades:
   a) Seguridad e inanición.
   b) Viveza y seguridad.
   c) Interbloqueo e inanición.
   d) Exlusión mútua y viveza.

2) La exclusión mutua mediante inhibición de interrupciones:
   a) Garantiza la ausencia de inanición.
   b) Mejora el rendimiento de las aplicaciones
   c) **No puede utilizarse en sistemas multiprocesador**.
   d) Únicamente garantiza la exclusión mutua en operaciones de E/S.

3) El algoritmo de exclusión mutua de Dekker:
   a) Está orienado a entornos distribuidos.
   b) Está orientado a entornos centralizados
   c) Es válido para n procesos con apenas modificaciones.
   d) Presenta situaciones en las que puede no garantizar las propiedades de viveza.

4) El problema de interbloqueo:
   a) Solo tiene solución si se resuelve mediante el uso de semáforos.
   b) Solo tiene solución si se resuelve mediante el uso de monitores.
   c) Solo tiene solución si se resuelve mediante el uso de algoritmos de espera ocupada.
   d) Todas las anteriores son falsas.

5) En términos de eficiencia:
   a) Los algoritmos de espera ocupada son más eficientes que los semáforos.
   b) Los monitores son más eficientes que los semáforos.
   c) A priori, no puede determinarse qué técnica de sincronización es la más eficiente.
   d) La eficiencia de los semáforos depende exclusivamente de la CPU.

6) La operación wait(s):
   a) Bloquea el proceso que la ejecuta si s=1.
   b) Bloquea el proceso que la ejecuta si s=0;
   c) Decrementa el valor de s y entonces bloquea el proceso si s=0.
   d) Si s=0 decrementa el valor de s y bloquea el proceso.

7) La gestión de los procesos bloqueados en un semáforo:
   a) **Debe ser siempre FIFO para evitar la inanición**.
   b) Puede ser FIFO o LIFO
   c) El Sistema Operativo desbloqueará los procesos en función de la prioridad.
   d) Mediante el uso de semáforos, los procesos no pasan a estado bloqueado.

8) Un semáforo **"s"** inicializado al valor 2
   a) Permite que dos procesos estén simultáneamente en su sección crítica.
   b) Dos procesos podrán ejecutar wait(s) sin bloquearse.
   c) Los semáforos se inicializan siempre a valor 1.
   d) El primer proceso que alcance la sentencia wait podrá acceder a su sección crítica.

9) Los monitores requieren de la utilización y definición de dos tipos de procesos:
   a) Procesos bloqueados y procesos bloqueantes
   b) **Procesos activos y procesos pasivos**.
   c) Procesos padre y procesos hijo.
   d) Proceso monitor y proceso principal.

10) En los monitores los procesos bloqueados:
   a) Se bloquean en las colas asociadas a variables de condición.
   b) Se bloquean en las colas de acceso al propio monitor
   c) Solo pasan a estado bloqueado si se ejecuta *delay*(var)
   d) a y b son ciertas

11) En la semántica resume & exit, el proceso desbloqueado por *resume(v)* es:
   a) El primer proceso que estuviera bloqueado en la cola de la variable de condición *v*
   b) El primer proceso que estuviera esperando para acceder al monitor.
   c) El sistema elige aleatoriamente entre la alternativa a y b
   d) Ninguna de las anteriores es cierta.

12) En el direccionamiento asimétrico del paso de mensajes:
   a) El emisor no identifica al receptor pero el receptor identifica al emisor
   b) El emisor identifica al receptor, pero el receptor no identifica al emisor
   c) El emisor identifica al receptor y el receptor identifica al emisor.
   d) El emisor no identifica al receptor y el receptor no identifica al emisor

13) En la instrucción de espera selectiva select, el proceso que la ejecuta se bloquea si:
   a) No se cumple ninguna de las guardas, si las tuviera.
   b) No disponga de alternativa else
   c) No existe ningún mensaje en los buzones/canales que se manejan
   d) La instrucción select no genera bloqueo del proceso.

14) El paso de mensajes síncrono permite la comunicación:
   a) Uno a uno.
   b) Uno a muchos
   c) Muchos a muchos
   d) Muchos a uno.

15) En las llamadas a procedimiento remoto (RPC), la invocación al resguardo del cliente:
   a) Siempre genera el bloqueo del proceso que realiza la invocación.
   b) Debe garantizar que existe concordancia entre los parámetros.
   c) No requiere de conexión entre cliente y servidor.
   d) La invocación se realiza siempre a un módulo que se encuentra en otro sistema.

---

1. Enunciar las condiciones de Bernstein (**1 punto**)
2. Discutir la corrección o incorrección del siguiente algoritmo de exclusión mutua (**1 punto**)

process P_0	process P_1

Repeat
C0=quiereentrar;
while turno ≠ 0 do
while C1=quiereentrar do;
turno=0;
Sección Critica0;
C0=restoproceso;
Resto0;
Forever
	Repeat
	C1=quiereentrar;
	while turno ≠ 1 do
	while C0=quiereentrar do;
	turno=1;
	Sección Critica1;
	C1=restoproceso;
	Resto1;
	Forever

---

3. Supongamos que un proceso productor y **k** procesos consumidores comparten un buffer limitado de **n** elementos. El productor deposita elementos en el buffer mientras haya sitio y los consumidores los extraen. Cada mensaje depositado por el productor debe ser recibido por los **k** consumidores. Más aún, cada consumidor recibe los mensajes en el orden en que fueron depositados; sin embargo, diferentes consumidores pueden recibir los mensajes en tiempos diferentes. Resolver el problema usando paso de mensajes asíncrono (**2,5 puntos**).

4. Hay tres clases de procesos que acceden a una lista FIFO compartida: buscadores, insertores y extractores, y cuya sintaxis es la siguiente:
   - Insertar (lista, elem); //Inserta elem al final de la lista
   - Extraer (lista); //Borra el primer elemento de la lista
   - Buscar(lista,elem): boolean; // Devuelve true si elem está en la lista

   Los buscadores simplemente examinan la lista y se pueden ejecutar concurrentemente con cualquier otro buscador. Los insertores son mutuamente excluyentes entre ellos, al igual que los borradores. Además no se puede insertar y borrar a la vez. Implementar una solución este problema de sincronización mediante semáforos. (**1,5 puntos**).

5. Dado el problema anterior, discutir una solución mediante monitores, suponiendo que los insertores son excluyentes entre sí, pero no con los borradores; y de igual modo, los borradores son excluyentes entre sí pero no con los insertores. (**1 punto**).

6.

Ejercicio 3

Debe considerarse que en el paso de mensajes asíncrono, el uso de buzones funciona como un buffer que es manejado en orden FIFO. Diseñaremos tres tipos de procesos: productor, consumidor y controlador. Usaremos un único buzón en el que el productor depositará los mensajes en el orden de producción. El controlador recogerá el mensaje y lo depositará en los k buzones (de tamaño usados para comunicarse con cada proceso consumidor. Dado que el buzón (buffer) tiene tamaño n, no puede garantizarse que cada consumidor haya retirado todos los mensajes que le han sido remitidos antes de que el productor haya producido un nuevo elemento. Para forzar la retirada de mensajes, cada consumidor envía un mensaje cuando ha retirado un mensaje de su buzón (solo si estaba lleno). De esa forma, el controlador debe quedar bloqueado hasta que haya hueco en el buzón y pueda depositar el mensaje.

```
Process Productor();
Var
        elem:tipoElemento;
Begin
        Repeat
                elem=newTipoElememto();
                send (buzonProductor,elem);
        forever
end;
```

```
Process Consumidor (id:entero)
Var
        elem:tipoElemento;
Begin
        Repeat
                If full(buzonConsumidor[id]) then  //Solo notificaremos al controlador que ya hay hueco si el buzón estaba lleno
                        receive(buzonConsumidor[id],elem); // retiramos un mensaje del buzón
                        send(haySitio[id],1); //Mandamos cualquier cosa para avisar de que ya hay sitio en el buzón
                else
                        receive(buzonConsumidor[id],elem); //
                consumir(elem);
        forever
end;
```

```
Process Controlador();
Var
        bufferPendientes=array[1..nCons] of tipoElemento; //array para almacenar mensajes pendientes de enviar
        elem:tipoElemento;
        i,j,k:entero
Begin
        repeat
                select
                        receive (buzonProductor,elem)
                        for i=1 to nCons do
                                if not full(buzonConsumidor[i] then
                                        send(buzonConsumidor[i],elem);
                                //Si buzonConsumidor[i] está lleno, se bloquearía el proceso
                                else
                                        bufferPendientes[i].i=elem; //Almacenamos en el buffer de pendientes el mensaje
                or
                        for j=1 to nCons replicate
                                receive (haySitio[j],k); // Consultamos los buzones que hayan dejado hueco.
                                send(buzonConsumidor[j],bufferPendientes[j]); //Enviamos el mensaje pendiente;
                endSelect
        forever
end;
```

```
Programa Prod_Cons
Const
        N=….
        nCons=….
Var
        buzonProductor=mailbox[1..N] of tipoElemento;
        buzonConsumidor=array[1..nCons] of tipoElemento;
        haySitio=array[1..nCons] of entero;
```

```
        i=entero;
Begin
        cobegin
                productor();
                controlador();
                For =i=1 to nCons do consumidor(i);
        coend

End.
```

Ejercicio 4

Usaremos dos semáforos. El primero controlará el acceso a la sección crítica de los procesos de inserción y borrado. El segundo controlará el número de buscadores que pretenden buscar, con el fin de que una vez que un buscador acceda a la sección crítica, finalice la búsqueda. Opcionalmente se ha añadido un semáforo – *vacio* – para garantizar que la primera operación que se realiza es una inserción.

```
Process insertar();
Begin
        ….
        wait (sem_insertar_borrar);
        insertar(lista,elem);
        signal (sem_insertar_borrar);
        signal(vacio); // indicar que la lista no está vacía
        …..
end;
```

```
Process borrar ()
Begin
        wait(vacio); // No se puede borrar si la lista está vacía
        wait (sem_insertar_borrar)
        Extraer (lista);
        Signal (sem_insertar_borrar);
        ….
End;
```

```
Process consultar
Begin
        ….
        // No es relevante controlar la lista vacía o no para los buscadores, ya que si está vacía no se encontrará el elemento
        // Si quisiera controlarse, bastaría con hacer wait (vacio)
        wait (sem_numBuscadores); //Para exclusión mutua de numBuscadores
        numBuscadores++;
        if numBuscadores>1 then
                wait(sem_insertar_borrar); // Si hay buscadores esperando, bloqueo a los insertores y extractores
        signal(sem_numBuscadores);
        está=buscar(lista,elem);
        wait (sem_numBuscadores); //Para exclusión mutua de numBuscadores
        numBuscadores--;
        if numBuscadores==0 then
                signal(sem_insertar_borrar); // Si ya no hay buscadores esperando, desbloqueo a los insertores y extractores
        signal(sem_numBuscadores);
        ….
end;
```

```
Program ejercicio4
var
        sem_numBuscadores,      //semáforo para controlar acceso controlado al contador de número de buscadores
        sem_insertar_borrar,    //semáforo para controlar acceso a inserción/borrado
        vacio: semaphore;       // semáforo para forzar que la primera ejecución sea de inserción o búsqueda
begin
        initial(sem_numBuscadores,1);
        initial(sem_insertar_borrar,1);
        initial(vacio,0);
```

```
        cobegin
                insertar();
                borrar();
                consultar();
        coend;
end.
```

Ejercicio 5

Debe tenerse en cuenta que el acceso a las operaciones del monitor se ejecutan en exclusión mutua. Por consiguiente si se encapsulara cada una de las operaciones a sincronizar en un procedimiento del monitor, todas ellas serían excluyentes entre sí. Por ello, la solución propuesta de seguir la filosofía de solicitar permiso/testigo para realizar cada una de las operaciones. Usaremos dos testigos: uno para el bloqueo de los insertores por parte de los borradores y otro para el bloqueo de los procesos borradores por parte de los insertores. No se requiere uso de monitor en los buscadores, ya que de incluirse en el monitor se ejecutaría en exclusión mútua con todos los demás procesos. La mejor opción sería, pues, dejar fuera del monitor la operación buscar.

```
Monitor Controlador;
Export
        quieroInsertar, yaHeInsertado, quieroBorrar, yaHeBorrado, quieroBuscar, yaHeBuscado;
var
        bloqueadoBorrar, bloqueadoInsertar:boolean;        //Controlará si se está insertando o borrando
        colaInsertar, colaBorrar, colaBuscar: condition;
        nEspInsertar, nEspBorrar, nEspBuscar: entero;
```

```
Procedure quieroInsertar ();
begin
        nEspInsertar++;
        if bloqueadoBorrar then delay (colaInsertar); // Si se está borrando, se bloquea
        else bloqueadoInsertar=TRUE; //bloquea el acceso a los borradores
        resume (colaInsertar); // Desbloqueo en cascada a un insertor que estuviera bloqueado
end;
```

```
Procedure ya HeInsertado();
Begin
        nEspInsertar--;
        if (nEspInsertar=0) resume (colaBorrar); //Desbloqueo de un borrador que estuviera esperando
end;
```

```
Procedure quieroBorrar (); //Idéntico a los anteriores pero cambiando  las variables usadas
begin
        nEspBorrar++;
        if bloqueadoInsertar then delay (colaBorrar); // Si se está insertando, se bloquea
        else bloqueadoBorrar=TRUE; //bloquea el acceso a los insertores
        resume (colaBorrar); // Desbloqueo en cascada a un borrador que estuviera bloqueado
end;
```

```
Procedure ya HeBorrado();
Begin
        nEspBorrar--;
        if (nEspBorrar=0) resume (colaInsertar); //Desbloqueo de un insertor que estuviera esperando
end;
```

```
Begin
        nEspInsertar=0; nEspBorrar=0;
        bloqueadoInsertar=FALSE;
        bloqueadoBorrar=TRUE; //Así se garantiza que la primera operación a realizar es una inserción
end.
```