Examen Sistemas Concurrentes y Distribuidos
10 de Julio de 2012

Apellidos, Nombre:
Grupo (M/T):

1) A continuación tiene 27 preguntas con cuatro posibles respuestas cada una. Por cada pregunta sólo una de las cuatro respuestas es correcta. Cada pregunta acertada vale 0.26. Cada respuesta errada descuenta un tercio del valor de una pregunta acertada. Las preguntas no contestadas no suman ni restan. Se ruega que rellene la siguiente tabla con sus respuestas, sólo se corregirán las respuestas que estén indicadas en dicha tabla. (7puntos)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
19	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**									

1. Desde el punto de vista de un Sistema Operativo un proceso es:
   a) **Entidad lógica a la que la CPU podrá planificar y asignar recursos.**
   b) Entidad lógica que podrá ser cargada en memoria para su planificación.
   c) Entidad lógica que se almacena en un dispositivo de almacenamiento.
   d) Ninguna de las anteriores es correcta.

2. La ejecución concurrente de varios procesos implica:
   a) La necesidad de múltiples unidades de procesamiento.
   b) Que existan múltiples programas dentro del sistema.
   c) **Una arquitectura del Sistema Operativo que la permita.**
   d) Un sistema Operativo Monoprogramado.

3. Para un correcto funcionamiento de los procesos concurrentes se debe asegurar:
   a) La exclusión mutua y la sincronización.
   b) Sólo la exclusión mutua.
   c) La exclusión mutua, la sincronización y evitar el interbloqueo.
   d) **Ninguna de las anteriores es correcta.**

4. La relación existente entre procesos e hilos es:
   a) **Los hilos están asociados al proceso que los crea.**
   b) El Sistema Operativo debe manejar la misma información que para el mantenimiento de los procesos.
   c) Los recursos podrán ser asociados tanto a los procesos como a los hilos.
   d) Los procesos son estructuras *ligeras* mientras que los hilos son estructuras *pesadas*.

5. La posibilidad que nos permite un sistema multihilo es:
   a) No ofrece ninguna ventaja sobre un sistema multiproceso.
   b) **Permite una mejor paralización de un problema sin necesidad de crear nuevos procesos.**
   c) Son un elemento presente en todos los Sistemas Operativos.
   d) Ninguna de las anteriores es correcta.

6. Para poder seguir la ejecución de un hilo será necesario almacenar:
   a) Una cantidad de información similar a la necesaria para gestionar un proceso.
   b) **Al menos la información de contexto y pila.**
   c) La información de contexto, pila y recursos asignados.
   d) Ninguna de las anteriores es correcta.

7. La exclusión mutua entre diferentes procesos garantiza:
   a) **El acceso seguro a la información compartida entre procesos.**
   b) No es necesario garantizar la exclusión mutua entre procesos.
   c) Sólo es necesaria en Sistemas Distribuidos.
   d) El acceso seguro a los recursos compartidos.

8. El algoritmo de Dekker:
   a) Soluciona el problema de sincronización entre procesos.
   b) Es un algoritmo incorrecto para la solución de la exclusión mutua.
   c) **Soluciona mediante espera ocupada el problema de la exclusión mutua**.
   d) Sufre de inanición para el problema de la exclusión mutua.

9. El algoritmo de Peterson frente al de Dekker:
   a) Tiene una mejor solución para el problema de sincronización entre procesos.
   b) No tiene el problema de espera ocupada que sí tiene el de Dekker.
   c) Es más eficiente que el algoritmo de Dekker.
   d) **Ninguna de las anteriores es correcta.**

10. Los semáforos son:
    a) Herramientas que solucionan el problema de la exclusión mutua.
    b) Herramientas para el problemas de concurrencia en Sistemas Distribuidos.
    c) **Una estructura de datos con operaciones atómicas para su manejo**.
    d) Ninguna de las anteriores es correcta.

11. La inicialización de la variable de un semáforo:
    a) **Sólo puede hacerse una única vez en su ciclo de vida**.
    b) No se iniciativa en un ciclo de vida.
    c) Puede inicializarse tantas veces como se quiera.
    d) Ninguna de las anteriores es correcta.

12. La operación **signal(..)** de un semáforo:
    a) Incrementará siempre el valor de la variable del semáforo.
    b) No hará nada a la variable del semáforo.
    c) Si hay procesos bloqueados no incrementará el valor de la variable del semáforo.

d) **Ninguna de las anteriores es correcta.**

13. Los monitores en relación a los semáforos:
    a) Son herramientas de más bajo nivel de programación.
    b) **Son herramientas de más alto nivel de programación con una estructura que ayuda a la corrección del programa.**
    c) No ayudan más que los semáforos.
    d) Ninguna de las anteriores es correcta.

14. La característica principal de un monitor es:
    a) **Todas las funciones se ejecutan en exclusión mutua**.
    b) Solucionan el problema de la sincronización entre procesos concurrentes.
    c) Sólo hay un proceso en el monitor en cada momento.
    d) Ninguna de las anteriores es correcta.

15. Las variables de condición en un monitor:
    a) Garantizan la exclusión mutua de las funciones del monitor.
    b) **Son necesarias para poder mantener la sincronización de los procesos dentro del monitor**.
    c) Son como los semáforos dentro del monitor.
    d) Controlan diferentes condiciones dentro del monitor.

16. El paso de mensajes entre procesos es necesario para:
    a) **El correcto funcionamiento entre procesos en un Sistema Distribuido**.
    b) El correcto funcionamiento entre procesos dentro de los Sistemas Concurrentes.
    c) Permite intercambiar información entre procesos.
    d) Soluciona el problema de la exclusion mutua entre procesos en un Sistema Distribuido.

17. En la comunicación directa entre procesos es necesario:
    a) Conocer el destinatario del mensaje.
    b) Conocer el remitente del mensaje.
    c) No se requiere ningún tipo de identificación.
    d) **El emisor debe conocer al destinatario y el receptor al remitente**.

18. En la comunicación asíncrona entre procesos:
    a) El buffer sólo se comparte entre emisor y receptor.
    b) No ha necesidad de buffer en la transmisión.
    c) El tamaño de buffer debe especificarse en la comunicación.
    d) **Ninguna de las anteriores es correcta**.

19. En la comunicación asíncrona entre procesos:
    a) **La primitiva de recepción bloqueará al proceso si no hay datos en en buzón**.
    b) La primitiva de envío bloqueará el emisor.
    c) Ninguna primitiva de envío o recepción bloquearán a los procesos implicados.
    d) Ambas primitivas de envío o recepción bloquearán a los procesos implicados.

20. En el problema del productor/consumidor si la primitiva de envío no bloquea al productor:
    a) El emisor deberá asegurarse que el consumidor esté disponible.
    b) **Deberemos utilizar un buzón de tamaño indefinido**.
    c) No hay solución posible con esa suposición de partida.
    d) Ninguna de las anteriores es correcta.

21. En la comunicación síncrona entre procesos:
    a) El emisor espera siempre al receptor antes de iniciar la transmisión.
    b) El receptor espera siempre al emisor antes de iniciar la transmisión.
    c) Ni emisor ni receptor esperan antes de iniciar la transmisión.
    d) **El primero que alcanza la primitiva de comunicación deberá esperar hasta que el otro alcance la suya antes de iniciar la transmisión**.

22. La utilización de un canal:
    a) **Establecerá el tipo de información que se transmitirán emisor y receptor en una comunicación síncrona**.
    b) Establecerá el tipo de sincronización necesaria en la comunicación.
    c) Permitirá el almacenamiento de información para la comunicación entre procesos.
    d) Ninguna de las anteriores es correcta.

23. La utilización de un canal de sincronización:
    a) **Se utilizará como elemento de sincronización entre procesos en entornos remotos**.
    b) Permite definir un tipo por defecto en la comunicación síncrona.
    c) No existe ese tipo de canales.
    d) Es el tipo de canales habituales en las comunicaciones sínconas.

24. La llamada a procedimiento remoto:
    a) **Permite la ejecución de un procedimiento presente en un proceso remoto dentro de un Sistema Distribuido**.
    b) Es un tipo de comunicación habitual en Sistemas Distribuidos.
    c) Es un elemento necesario en la estructura de los Sistemas Distribuidos.
    d) Ninguna de las anteriores es correcta.

25. Un proceso que invoca una llamada a procedimiento remoto:
    a) No esperará a la respuesta por parte del proceso remoto.
    b) **Desde el punto de vista del programador es trasparente como si utilizara una biblioteca perteneciente a su sistema**.
    c) Sólo implica una degradación de las prestaciones del proceso dentro del sistema.

d) El programador deberá conocer información relativa a la estructura del proceso remoto.

26. En el proceso de resolución de una llamada a procedimiento remoto:
    a) Los mensajes que han de transmitirse deberá confeccionarlos el programador.
    b) El programador deberá tener presente la codificación de la información el la máquina remota.
    c) **Es responsabilidad del sistema la solución a la transmisión de la información**.
    d) Ninguna de las anteriores es correcta.

27. En la llamada a procedimiento remoto:
    a) Los dos sistemas deberán tener una misma arquitectura.
    b) Deberá ser el mismo Sistema Operativo en las máquinas remotas.
    c) Se utilizará el mismo lenguaje de programación para codificar los procesos.
    d) **Ninguna de las anteriores es correcta.**

2) (1 punto) Tenemos un aparcamiento donde hay una capacidad para 500 coches. El acceso al aparcamiento es mediante una rampa donde sólo puede pasar un único coche para entrar o para salir del aparcamiento. Mientras haya coches en un sentido los coches del sentido contrario tendrán que esperar. Resolver el problema de sincronización mediante semáforos. La solución deberá estar adecuadamente justificada.

SOLUCIÓN:

Vamos a suponer que el programa principal lanzará los procesos correspondientes a los coches que quieren entrar en el parking y los que quieren salir del parking e inicializará las variables compartidas y semáforos a los valores que se indican más adelante:

Los semáforos a utilizar serán los siguientes:

exmEntrada: semáforo inicializado a 1 para gestionar la exclusión mutua de los procesos de los coches que quieren entrar al parking.

exmSalida: semáforo inicializado a 1 para gestionar la exclusión mutua de los procesos de los coches que quieren entrar al parking.

rampa: semáforo incializado a 1 que controla el acceso a la rampa del parking para decidir qué coches pueden entrar o salir del mismo.

capParking: semáforo inicializado a 500 que es la capacidad inicial del parking. Controlará la capacidad del mismo para evitar que entren más coches de los permitidos.

Las variables compartidas son:

cochesEntrada: variable entera inicializada a 0 que cuenta los coches que están entrando en el parking.

cochesSalida: variable entera inicializada a 0 que cuenta los coches que están saliendo del parking.

Proceso cochesEntrada

```
wait(capParking) \\ Hay que asegurarse que hay sitio en el parking

wait(exmEntrada) \\ Entrada en exclusión mutua
  cochesEntrada++;
  Si (cochesEntrada == 1) entonces
    wait(rampa) \\ Si es el primero que quiere bajar reserva la rampa
  finSi
signal(exmEntrada) \\ Fin exclusión mutua

procesoEntradaParking();

wait(exmEntrada) \\ Entrada en exclusion mutua
  cochesEntrada--;
  Si (cochesEntrada == 0) entonces
    signal(rampa) \\ Si es el último libera la rampa
  finSi
signal(exmEntrada) \\ Fin del proceso de entrada al parking
```

Proceso cochesSalida

```
wait(exmSalida) \\ Entrada en exclusión mutua
  cochesSalida++;
  Si (cochesSalida == 1) entonces
    wait(rampa) \\ Si es el primero que quiere salir reserva la rampa
  finSi
signal(exmSalida) \\ Fin exclusión mutua

procesoSalidaParking();

wait(exmSalida) \\ Entrada en exclusión mutua
  cochesSalida--;
  Si (cochesSalida == 0) entonces
    signal(rampa) \\ Si es el último libera la rampa
  finSi
signal(exmSalida) \\ Fin exclusión mutua

signal(capParking) \\ Libera una plaza del parking y finaliza la salida
```

3) (1 punto) Una cuenta de ahorros es compartida entre distintas personas (procesos). Cada persona puede sacar o depositar dinero en la cuenta. El balance actual de la cuenta es la suma de los depósitos menos la suma de las cantidades sacadas. El balance nunca puede ser negativo. Las soluciones deberán estar adecuadamente justificadas.

1. Construir un monitor para resolver este problema con las operaciones **depositar(cantidad)** y **extraer(cantidad)**.

2. Modificar la solución anterior suponiendo que las extracciones son resueltas por orden de llegada. Es decir, si en la cuenta hay 200 y alguien quiere sacar 300 y después llega otra persona para sacar 200, este último tendrá que esperar hasta que el de la petición de 300 consiga su cantidad.

SOLUCIÓN:
Una posible solución para el monitor podría ser:

```
monitor cuenta;

  export depositor, extraer;
  var
    balance,primera_peti : enteros;
    lista: lista FIFO de enteros;
    espera: variable de condición;

  procedure depositar (cantidad: entero) {

    balance = balance + cantidad;
    Si (NO vacia(lista)) entonces \\ Hay peticiones de extracción
      primera_peti = primero(lista); \\ La tenemos en cuenta
    siNo
      primera_peti = 0;
    finSi

    Si ((primera_peti<>0) Y (primera_peti<=balance) entonces
      \\ Si el que está esperando puede extraer lo desbloqueamos
      eliminar_elemento(lista);
      resume(espera);
    finSi
  }

  procedure extraer (cantidad: entero) {

    Si (cantidad>balance) entonces
      \\ Si no se puede extraer, nos bloqueamos
      insertar_elemento(lista,cantidad);
      delay(espera);
    finSi

    balance = balance - cantidad;

    Si (NO vacia(lista)) entonces \\ ¿Hay procesos bloqueados?
      primera_peti = primero(lista); \\ La tenemos en cuenta
    siNo
      primera_peti = 0;
    finSi

    Si ((primera_peti<>0) Y (primera_peti<=balance) entonces
      \\ Si el que está esperando puede extraer lo desbloqueamos
      eliminar_elemento(lista);
      resume(espera);
    finSi
  }

  {
    balance = 0;
    primera_peti = 0;
    inicializar(lista);
}
```

En el anterior monitor, la lista almacena la cantidad a extraer por cada proceso con orden FIFO. La operación `primero(lista)` devuelve el valor del primer elemento de la lista. La operación `eliminar_elemento(lista)` elimina el primer elemento de la lista.

Para el segundo apartado una posible solución pasa por modificar el procedimiento extraer:

```
  procedure extraer (cantidad: entero) {

    Si ((cantidad>balance)  O (NO vacia(lista)) entonces
      \\ Si no se puede extraer o peticiones pendientes, nos bloqueamos
      insertar_elemento(lista,cantidad);
      delay(espera);
    finSi

    balance = balance - cantidad;

    Si (no vacia(lista)) entonces \\ ¿Hay procesos bloqueados?
      primera_peti = primero(lista); \\ La tenemos en cuenta
    siNo
      primera_peti = 0;
    finSi

    Si ((primera_peti<>0) Y (primera_peti<=balance) entonces
      \\ Si el que está esperando puede extraer lo desbloqueamos
      eliminar_elemento(lista);
```

```
      resume(espera);
    finSi
  }
```

4) (1 punto) Supongamos que **n** procesos comparten tres impresoras. Antes de usar una impresora el proceso debe solicitar una impresora, devolviéndole el sistema el código de la impresora asignada. Una vez utilizada la impresora, ésta es liberada. Desarrollar una solución al problema empleando para ello el modelo de comunicación de canales. La solución deberá estar adecuadamente justificada.

SOLUCIÓN:

Una posible solución para el problema podría ser:

```
Constantes
  NumProcesos = 10;
  NumImpresorar = 3;

Variables
  peticionImpresora : array [1..NumProcesos] de canales de enteros;
  liberarImpresora: array [1..NumProcesos] de canales de enteros;
  contador : entero;

Proceso (ident: entero) {
  Variables
    idImpresora : entero;

  Repetir
    ...
    \\ Proceso necesita impresora
    peticionImpresora[ident] ? idImpresora;
    ....
    \\ Usar impresora idImpresora
    ....
    \\ Liberar impresora
    liberarImpresora[ident] ! idImpresora;
    .....
   Indefinidamente
 }

 Controlador {
   Variables
     idImpresoras : array [1..NumImpresorar] de enteros;
     cont1, cont2, numImpresorasLibres : enteros;
     imprsoraLibre: entero;

   Para cont = 1 hasta NumImpresoras hacer
     idImpresoras[cont1] = cont1;

   numImpresorasLibres = NumImpresoras;

   Repetir
     Select
       for cont1 = 1 to NumProcesos replicate
         when nunImpresorasLibres>0 =>
 \\ Supongamos que tenemos una función pedirImpresoraLibre
 \\ que analiza el vector idImpresoras y nos devuelve el identificador
 \\ de una impresora libre y la marca como ocupada.
           peticionImpresora[cont1] ! pedirImresoraLibre();
           numImpresorasLibres --;
     or
       for cont2= 1 to NumProcesos replicate
         liberarImpresora[cont2] ? impresoraLibre;
         numImpresorasLibres++;
         idImpresoras[impresoraLibre] = impresoraLibre;
     or
       terminate
     finSelect
   Indefinidamente
 }
```

Para la solución suponemos te tendremos un numero **NumProcesos** de procesos que han sido lanzado concurrentemente con su identificador correspondiente.