Examen Sistemas Concurrentes y Distribuidos
6 de Junio de 2013

Apellidos, Nombre:
Grupo (M/T):

1) A continuación tienes 18 preguntas con cuatro posibles respuestas cada una. Por cada pregunta sólo una de las cuatro respuestas es correcta. Cada pregunta acertada vale 0,2 puntos, cada respuesta errada descuenta 0,07 puntos y 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. (3,5 puntos)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**

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

2. ¿Qué son las condiciones de Bernsein?
   a) Indican si dos o más procesos pueden ejecutarse concurrentemente.
   b) **Determinan si un conjunto de instrucciones pueden ejecutarse concurrentemente.**
   c) Ayudan a la sincronización de los procesos.
   d) Sirven para determinar las secciones críticas de los procesos.

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

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

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

6. 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 procesos son estructuras *ligeras* mientras que los hilos son estructuras *pesadas*.
   d) Los recursos podrán ser asociados tanto a los procesos como a los hilos.

7. En la semántica **resume & exit**, el proceso desbloqueado por `resume(v)`:
   a) Se elige aleatoriamente procesos bloqueados en la variable o en el monitor.
   b) El primer proceso que estuviera esperando para acceder al monitor.
   c) Ninguna de las respuestas es correcta.
   d) **El primer proceso que estuviera bloqueado en la cola de la variable de condición `v`.**

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

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

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

11. La operación `signal(.)` de un semáforo:
    a) **Ninguna de las respuestas es correcta.**
    b) No hará nada con la variable del semáforo.
    c) Incrementará siempre el valor de la variable del semáforo.
    d) Si hay procesos bloqueados sólo no incrementará el valor de la variable del semáforo.

12. La inicialización de la variable de un semáforo:

a) Puede inicializarse tantas veces como se quiera.
b) No se inicializa el el ciclo de vida.
c) **Sólo puede hacerse una única vez en su ciclo de vida.**
d) Ninguna de las respuestas es correcta.

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

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

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

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

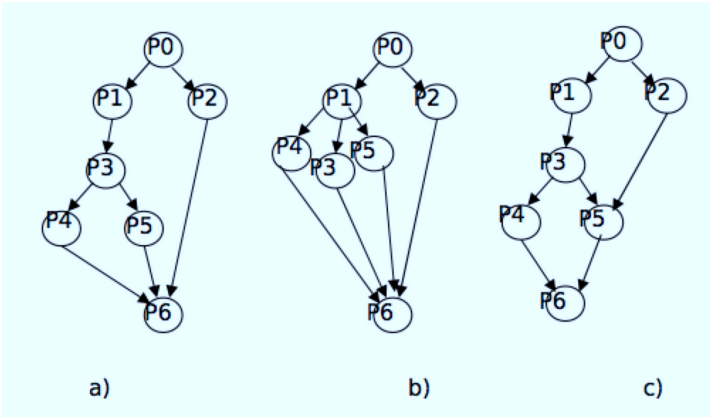
17. En la comunicación asíncrona entre procesos:
    a) No se requiere ningún tipo de identificación.
    b) No hay necesidad de buffer en la transmisión.
    c) El *buffer* sólo se comparte entre emisor y receptor.
    d) **Ninguna de las respuestas es correcta.**

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

1) (1 punto) Concepto de semáforo. Similitudes y diferencias con las variables de condición de los monitores.

2) (1 punto) Ventajas e inconvenientes entre las posibilidades que ofrece el paso de mensajes asíncronos frente a los sincronos.

3) (1,5 puntos) Construir, utilizando las instrucciones concurrentes `COBEGIN-COEND` los programas concurrentes que se correspondan con los grafos de precedencia que se muestran a continuación:



Solución:

a
```
BEGIN
    P0
    COBEGIN
      P2
      BEGIN
        P1
        P3
        COBEGIN
          P4
          P5
        COEND
      END
    COEND
    P6
END
```

b

```
BEGIN
    P0
    COBEGIN
    P2
    BEGIN
      P1
      COBEGIN
        P3
        P4
        P5
      COEND
    END
    COEND
    P6
  END
```

c

```
BEGIN
    COBEGIN
      P2
      BEGIN
        P1
        P3
      END
    COEND
    COBEGIN
      P4
      P5
    COEND
    P6
  END
```

4) (1,5 puntos) Tenemos un sistema con `t` terminales e `i` impresoras. En cada terminal se ejecuta un proceso que usa recursos propios y accede a utilizar alguna de las `i` impresoras (siendo las `i` impresoras recursos comunes a los `t` terminales). Utilizar un monitor para que el acceso a una impresora sea en exclusión mutua pero pudiendo haber varios procesos a la vez en las impresoras, cada uno en una distinta. Resolver el ejercicio razonando la solución propuesta.

Solución

```
Process Terminal(t)

Begin
  // Sentencias antes de imprimir
  idImp = Impresoras.reservarImpresora;
  realizarImpresíon(idImp);
  Impresoras.finalizarImpresion(idImp);
  // Sentencias después de imprimir
End

Monitor Impresoras

Var
  impresoraLibre : [1..i] of boolean;
  numImprOcupadas : integer;
  esperaImpresora : condition;

Export
  reservarImpresora, finalizarImpreson;

function reservarImpresora : integer;

Begin
   If (numImprOcupadas = i)
      delay(esperaImpresora);
   EndIf
   idImp = 1;
   While (Not impresoraLibre[idImp])
     idImp++;
   numImprOcupadas++;
   impresoraLibre[idImp] = false;
   return idImp;
End

procedure finalizarImpresion ( idImp : integer );

Begin
  impresoraLibre[idImp] = true;
  numImprOcupadas--;
  resume(esperaImpresora);
End
```

```
Begin
  For j = 1 to i to
    impresoraLibre[j] = true; // impresora libre
  EndFor
  numImprOcupadas = 0
End
```

5) (1,5 puntos) Supongamos tres procesos: Pa, Pb y Pc de tal forma que Pa sólo puede escribir la letra 'a', Pb la letra 'b' y Pc la letra 'c'. Los tres procesos están en un bucle de 1 a 5. Se pide sincronizar los tres procesos para que se cumplan las siguientes condiciones:

1. Una 'b' siempre se imprime antes que una 'c'.
2. Las b's y las c's deben alternarse en la cadena de salida, de tal forma que después de la primera 'b' se ha impreso, ninguna otra 'b' puede ser impresa hasta que no se imprima una 'c'. De igual forma, cuando se imprime una 'c' no puede volver a imprimir otra hasta que no se imprima una 'b'.
3. El número total de b's y c's presentes en la cadena de salida en un instante determinado no puede exceder el numero de a's en la cadena de salida en ese mismo instante.

   La solución será programada mediante el paso de mensajes asíncronos. Resolver el ejercicio razonando la solución propuesta.

Solución

Para esta solulución vamos a suponer que sólo hay un proceso de cada tipo y de esta forma sólo será necesario un buzón para la sincronización con cada uno de ellos por parte del controlador.

```
Process Pa
Begin
  For i = 1 to 5 do
    msg = "A";
    Send (ImprimirA, msg);
    Receive (ImprimirA, msg); //Para poder imprimir la siguiente letra
    Imprimir(msg);
  EndFor
End

Process Pb
Begin
  For i = 1 to 5 do
    msg = "B";
    Send (ImprimirB, msg);
    Receive (ImprimirB, msg); //Para poder imprimir la siguiente letra
    Imprimir(msg);
  EndFor
End

Process Pc
Begin
  For i = 1 to 5 do
    msg = "C";
    Send (ImprimirC, msg);
    Receive (ImprimirC, msg); //Para poder imprimir la siguiente letra
    Imprimir(msg);
  EndFor
End

Process Controlador
Begin
  numA = numB = numbC = 0;
  ordenBC = "B";
  Do
    Select
        Receive (ImprimirA, msg);
        numA++;
        Send (ImprimirA, msg);
    Or
      When (numA > numB) AND (ordenBC = "B") =>
        Receive (ImprimirB, msg);
        numB++;
        ordenBC = "C";
        Send (ImprimirB, msg);
    Or
      When (numA > numC) AND (ordenBC = "C") =>
        Receive (ImprimirC, msg);
        numC++;
        ordenBC = "B";
        Send (ImprimirB, msg);
    End
  While (true)
End
```