Examen Sistemas Concurrentes y Distribuidos
26 de Mayo 2017

Apellidos, Nombre:
Grupo (M/T):

Preguntas de Respuesta Múltiple

A continuación tienes 20 preguntas con cuatro posibles respuestas cada una. Por cada pregunta solo una de las cuatro respuestas es correcta. Cada pregunta acertada vale 0,2 puntos, cada respuesta errada descuenta 0,05 puntos y las preguntas no contestadas no suman ni restan. Se ruega que rellene la siguiente tabla con sus respuestas, solo se corregirán las respuestas que estén indicadas en dicha tabla. (4 puntos)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**

1. 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

2. 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**

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

4. La exclusión mutua entre diferentes procesos garantiza:
   a) **Que solo un proceso puede estar dentro de la sección crítica**
   b) No es necesario garantizar la exclusión mutua entre procesos
   c) Sólo es necesaria en Sistemas Distribuidos
   d) El acceso seguro a todos los recursos de un proceso

5. En los programas concurrentes:
   a) Podemos determinar de forma clara el orden de ejecución de las diferentes instrucciones que lo componen
   b) El tiempo empleado para terminar la ejecución siempre es la misma
   c) **Se pueden producir resultados diferentes para el mismo conjunto de datos de entrada**
   d) Ninguna de las respuestas es correcta

6. 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

7. En el problema del **productor/consumidor** resuelto mediante semáforos:
   a) Los procesos productores deben sincronizarse entre sí para garantizar la corrección del problema
   b) **Los procesos productores deben sincronizarse con los procesos consumidores para garantizar la corrección del problema**
   c) Sólo es necesario garantizar la exclusión mutua al buffer compartido
   d) Ninguna de las respuestas es correcta

8. El problema del interbloqueo:
   a) Se resuelve mediante el uso de semáforos
   b) Se resuelve mediante el uso de monitores
   c) No es un problema que se da en la programación concurrente
   d) **Ninguna de las respuestas es correcta**

9. La sentencia `resume` de un monitor:
   a) Tiene la misma lógica de funcionamiento que la operación `signal` de un semáforo
   b) Permite bloquear a un proceso en el monitor dentro de una variable de condición
   c) Solo se aplica a una variable de condición del monitor si hay procesos bloqueados en la misma
   d) **Libera a un proceso bloqueado en la variable de condición del monitor. Si no hay, no tiene efecto**

10. La operación `wait(s)`:
    a) Bloquea el proceso que la ejecuta si `s=1`
    b) **Bloquea al 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

11. Un semáforo cuya variable se inicializa a 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 ejecute la sentencia `wait` podrá acceder a su sección crítica

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

13. 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) Podemos tener múltiples procesos bloqueados dentro del monitor
    d) **Todas las respuestas son correctas**

14. 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**

15. 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) **La comunicación entre procesos tiene que ser asíncrona**
    c) No hay solución posible con esa suposición de partida
    d) La comunicación entre procesos tiene que ser síncrona

16. 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 respuestas es correcta

17. 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 en la máquina remota
    c) **Es responsabilidad del sistema la solución a la transmisión de la información**
    d) Ninguna de las respuestas es correcta

18. Cual de las siguientes cuestiones han de resolverse en una llamada a procedimiento remoto
    a) La ejecución en espacios de direcciones de memoria diferentes
    b) El paso de parámetros
    c) La respuesta ante fallos de una máquina
    d) **Todas las respuestas son válidas**

19. En RPC asíncrona:
    a) a llamada a procedimiento bloquea al proceso cliente
    b) **la llamada a procedimiento no bloquea al proceso cliente**
    c) La resolución a la RPC se bloquea en servidor
    d) También es conocida como RPC síncrona extendida

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

Preguntas de Desarrollo Corto

1) (1 punto) Indicar las diferencias entre soluciones software y hardware en la solución del problema de **exclusión mutua**.

2) (1 punto) ¿Cuál sería el principal inconveniente de disponer de un esquema de comunicación mediante paso de mensajes y no disponer de la sentencia de espera selectiva? Razona la respuesta.

Problemas

1) (1 punto) Tenemos una cuenta compartida para la organización del viaje de fin de curso. En la cuenta se realizarán ingresos y retiradas. Las retiradas de la cuenta se realizarán en orden FIFO de peticiones, es decir, siempre se pagará primero al que primero realice la solicitud de pago, independientemente del dinero que haya en la cuenta. Para la solución se utilizarán semáforos.

Solución

```
Variables compartidas:
    exm : semáforo inicializado a 1
    retiradas : semáforo inicializado a 0
    saldo : variable entera
    pendientes : TDA FIFO de enteros
    retiradaPendiente : entero inicializado a 0
      para indicar que se a liberado un proceso de retirada

proceso Ingresos(id)
    wait(exm)
      saldo = saldo + cantidad

      if ( (saldo >= pendientes.head()) AND (NO pendientes.isEmpty()) ) {
        pendientes.remove()
        retiradasPendientes = retiradasPendientes + 1
        signal(retiradas)
      }
    signal(exm)

proceso Retiradas(id)
    wait(exm)
      if ( (pendientes.isEmpty()) AND (cantidad <= saldo)
            AND (retiradasPendientes == 0) ) {
        saldo = saldo - cantidad
      } else {
        pendientes.append(cantidad)
        signal(exm)
        wait(retiradas)

        wait(exm)
          saldo = saldo - cantidad
          retiradasPendientes = retiradasPendientes - 1
          if ( (saldo >= pendientes.head()) AND (NO pendientes.isEmpty()) ) {
            pendientes.remove()
            retiradasPendientes = retiradasPendientes + 1
            signal(retiradas)
          }
      }
    signal(exm)
```

2) (1 punto) Consideremos el problema del **productor/consumidor** con varios productores y varios consumidores donde el buffer compartido es un buffer de enteros de tamaño máximo para 16 elementos. Los consumidores extraen un entero cada vez que acceden al buffer, pero los productores pueden insertar más de un elemento a la ves, incluso el máximo del buffer, y deben quedar bloqueados hasta que haya suficiente espacio disponible en el buffer. Las peticiones de los productores deben ser atendidas en orden FIFO. Realizar el monitor que resuelve el problema.

Solución

```
monitor ProductorConsumidor

Variables
  buffer : TDA FIFO de enteros
  tamaño : tamaño del buffer (16)
  elementos : variable entera inicializada a 0
  productor : variable condición
  consumidor : variable condición
  esperaEspacio : TDA FIFO de enteros

Procedimientos públicos del monitor
  void añadir(elmProductor) añade una cantidad N de enteros al buffer
  entero retirar() retira un elemento del buffer

void añadir ( elmProductor ) {
  if ( ((elmProductor.size() + elementos) > tamaño)
       OR (NO emty(productor)) ) {
    esperaEspacio.append(elmProductor.size())
    delay(productor)
  }

  buffer.append(elmProductor)
  elementos = elementos + elmProductor.size()
```

```
  if ( (esperaEspacio.head() + elementos) <= tamaño ) {
    esperaEspacio.remove()
    resume(productor)
  } else {
    resume(consumidor)
  }
}

entero retirar() {
  if ( elementos == 0 ) {
    delay(consumidor)
  }

  elementos = elementos - 1
  elmConsumidor = buffer.get()

  if ( (esperaEspacio.head() + elementos) <= tamaño ) {
    esperaEspacio.remove()
    resume(productor)
  }
  return elmConsumidor
}
```

3) (1 punto) Tenemos un bloque de pisos de 6 plantas y un ascensor con un capacidad para 4 personas. Las peticiones de las personas se pueden realizar desde cualquier piso y serán para subir o para bajar a un piso determinado. Desarrollar el código para el proceso ascensor y el código para los procesos persona. Resolver el problema mediante paso de mensajes asíncronos.

NOTA: El ascensor empieza en la planta baja del edificio. Se atienden peticiones en el sentido que tenga el ascensor, es decir, si está subiendo no atiende a peticiones de bajada. El sentido cambia cuando no haya personas en el ascensor o llegue al final del trayecto. Si no hay peticiones y es ascensor está vacío se dirigirá a la planta baja.

Solución

Para la solución vamos a considerar que el piso 1 es la planta baja

```
Buzones de comunicación
  subir[piso] : buzón para solicitar subida
  bajar[piso] : buzón para solicitar bajada
  subirAscensor[id] : buzón para subir al ascensor
  dejarAscensor[id] : buzón para indicar que la persona se baja del ascensor

Variables
  capacidad : entero que indica el número de personas en el ascensor
  plantaAscensor : entero, planta en la que esta el ascensor inicial a 1
  direccion : indica la dirección del ascensor (subida/bajada)
              inicializada a subida
  mensaje : está compuesto (id proceso, planta de bajada)
  esperaSubir[piso] : TDA FIFO de mensajes
  esperaBajar[piso] : TDA FIFO de mensajes
  ocupantesAscensor : TDA FIFO de mensajes ordenado por planta y dirección

proceso PersonaSubida(id)
  enviar(subir[piso], mensaje)
  recibir(ascensor[id], any)
  recibir(dejarAscensor[id], any)

proceso Ascensor
  while ( true ) {
    select {
      for i = 1 to 5 replicate
        recibir(subir[i], mensaje)
        esperaSubir[i].append(mensaje)
    OR
      for i = 2 to 6 replicate
        recibir(bajar[i], mensaje)
        esperaBajar[i].append(mensaje)
    OR
      when (capacidad < 4) AND (NO esperaSubir[plantaAscensor].isEmpty())
           AND (direccion == subir)
      while ( ocupantesAscensor.head().planta == plantaAscensor ) {
        enviar(dejarAscensor[ocupantesAscensor.get().id], any)
        capacidad = capacidad - 1
      }
      while ( (capacidad < 4) AND
              (NO esperaSubir[plantaAscensor].isEmty() ) {
        ocupantesAscensor.append(esperaSubir[plantaAscensor].get())
        capacidad = capacidad + 1
      }
    OR
      when (capacidad < 4) AND (NO esperaBajar[plantaAscensor].isEmpty())
          AND (direccion == bajar)
      while ( ocupantesAscensor.head().planta == plantaAscensor ) {
        enviar(dejarAscensor[ocupantesAscensor.get().id], any)
        capacidad = capacidad - 1
      }
```

```
    while ( (capacidad < 4) AND
            (NO esperaBajar[plantaAscensor].isEmty() ) {
      ocupantesAscensor.append(esperaSubir[plantaAscensor].get())
      capacidad = capacidad + 1
    }
  OR
    when ocupantesAscensor.head().planta == plantaAscensor
    while ( ocupantesAscensor.head().planta == plantaAscensor ) {
      enviar(dejarAscensor[ocupantesAscensor.get().id], any)
      capacidad = capacidad - 1
    }

    if ( capacidad == 0 ) {
      if ( direccion == subir )
        direccion == bajar
      else
        direccion == subir
    }
  }

  if ( direccion == subir ) {
    plantaAscensor = plantaAscensor + 1
    if ( plantaAscensor = 6 )
      direccion = bajar
  } else {
    plantaAscensor = plantaAscensor - 1
    if ( plantaAscensor = 1 )
      direccion = subir
}
```

4) (1 punto) Un quiosco de venta de lotería se encuentra atendido por dos puestos A y B, y está organizado en forma de cola única. Cuando un cliente le llega el turno, se dirige al puesto que esté libre. Si ambos lo están, se dirige siempre al puesto A. Desarrollar el código necesario para resolver el problema mediante el paso de mensajes síncronos.

Solución

```
Canales de comunicación
  peticion[id] : se reciben las peticiones para la atención
  puesto[id] : se comunica el puesto de atención
  finalizacion[id] : se finaliza la atención del puesto

Variables
  esperaPuesto : TDA FIFO de enteros
  puestoA : variable booleana inicializada a libre
  puestoB : variable booleana inicializada a libre
  puestosLibres : variable entera inicializada a 2

proceso Persona(id)
  enviar(peticion[id], id)
  recibir(puesto[id], puesto)
  compraLoteria()
  enviar(finalizacion[id], puesto)

proceso Kiosko
  while( true ) {
    select {
      for i = 1 to numPersonas replicate
        recibir(peticion[i], id)
        esperaPuesto.append(id)
    OR
      when puestosLibres > 0
      if (puestoA == libre {
        puestoA = ocupado
        enviar(puesto[esperaPuesto.get()], A)
      } else {
        puestoB = ocupado
        enviar(puesto[esperaPuesto.get()], B)
      }
      puestosLibres = puestosLibres - 1
    OR
      for i = 1 to numPersonas replicate
        recibir(finalizacion[id], puesto)
        if ( puesto == A )
          puestoA = libre
        else
          puestoB = libre
        puestosLibres = puestosLibres + 1
     }
  }
```