EXAMEN-SCD-TEMA-1-Y-2--Resuelto-...

amorexp

Sistemas Concurrentes y Distribuidos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

EXAMEN SCD TEMA 1 Y 2

1. Señala la única opción correcta mediante un circulo. En caso de señalar una equivocadamente, marcar con una X la opción errónea.

- Sobre paralelismo y concurrencia.

a) El paralelismo real solo se puede dar en sistemas distribuidos.

b) En un sistema monoprocesador no puede implementarse ningún tipo de concurrencia.

c) El paralelismo real solo se puede dar si los procesos comparten una memoria común.

d) Todas las anteriores afirmaciones son falsas.

La afirmación correcta sería:

d) Todas las anteriores afirmaciones son falsas.

Veamos por qué:

a) El paralelismo real solo se puede dar en sistemas distribuidos: El paralelismo no está limitado exclusivamente a sistemas distribuidos. Los sistemas distribuidos pueden tener paralelismo, pero el paralelismo también se puede lograr en sistemas con múltiples núcleos o procesadores en una máquina.

b) En un sistema monoprocesador no puede implementarse ningún tipo de concurrencia: Aunque en un sistema monoprocesador el paralelismo es limitado, aún es posible implementar concurrencia a través de técnicas como la multitarea cooperativa o preemptiva, aunque no haya ejecución simultánea real debido a la limitación del hardware.

c) El paralelismo real solo se puede dar si los procesos comparten una memoria común: Si bien el paralelismo en sistemas compartiendo una memoria común puede ser más eficiente en términos de comunicación entre procesos, el paralelismo también puede ocurrir en sistemas con diferentes espacios de direcciones utilizando técnicas como la comunicación a través de mensajes (en lugar de compartir memoria).

- Considera la sentencia x:= x + 1, que se ejecuta repetidas veces por varios procesos concurrentes:

a) Cada vez que se ejecuta por cualquier proceso el tiempo que tarda podría ser distinto.

b) En sistemas monoprocesador, siempre tarda lo mismo en ejecutarse por cualquiera de los procesos.

c) En sistemas multiprocesador, siempre tarda lo mismo en ejecutarse por cualquiera de los procesos.

d) Siempre tarda lo mismo en ejecutarse por cualquiera de los procesos ya que es atómica.

La afirmación correcta sería:

a) Cada vez que se ejecuta por cualquier proceso el tiempo que tarda podría ser distinto.

La ejecución de una sentencia como x := x + 1 puede verse afectada por varios factores, incluso si se ejecuta en procesos diferentes:

Concurrencia y competencia por recursos: En sistemas concurrentes, varios procesos pueden competir por recursos como la CPU, la memoria, etc. Dependiendo de cómo se asignen estos recursos, el tiempo que tarda en ejecutarse una sentencia podría variar.

Planificación del sistema operativo: La planificación de los procesos por parte del sistema operativo también influye en el tiempo de ejecución. En un sistema multiprocesador, si hay varios procesos en ejecución, el sistema operativo puede programar la ejecución de cada proceso en diferentes núcleos o cambiar su prioridad, lo que puede afectar el tiempo de ejecución de la sentencia.

Acceso a caché y a memoria: El estado de la caché y la memoria también puede variar entre procesos, lo que puede influir en el tiempo de ejecución. Si un proceso tiene datos en la caché mientras que otro necesita acceder a los mismos datos desde la memoria principal, esto puede causar variaciones en el tiempo de ejecución.

En resumen, la ejecución de una sentencia no siempre tarda lo mismo cada vez que se ejecuta, especialmente en entornos concurrentes donde varios procesos compiten por recursos y la planificación del sistema operativo puede variar.

- Dado un programa concurrente compuesto por dos programas secuenciales A y B que se ejecutan concurrentemente donde el programa A se define como la secuencia de sentencias atómicas A1 -> A2 -> A3 y B como B1 -> B2 -> B3, se cumple que:

a) La secuencia de interfoliación B2 A1 A2 A3 B1 B3 es válida

b) La secuencia de interfoliación B1 A1 A2 B3 A3 B2 es válida

c) La secuencia de interfoliación A1 A2 B1 B2 A3 B3 es válida

d) Todas las anteriores son falsas

La secuencia de interfoliación describe el orden en el que las instrucciones de los programas A y B se ejecutan en un entorno concurrente.

La secuencia de interfoliación válida para que ambos programas se ejecuten concurrentemente debe tener coherencia con las dependencias de datos entre las instrucciones de cada programa. Vamos a analizar las opciones:

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

1/6
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

Si ya tuviste sufi con tanto estudio...
Te dejamos este espacio
para desahogarte.

Pinta, arranca,
llora... tú decides ;)

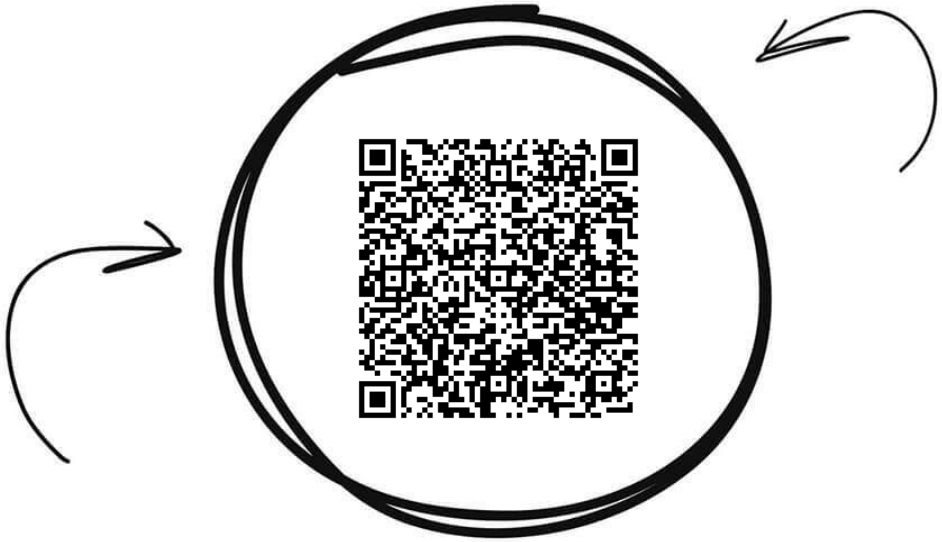
¿Te sientes más liberado?
Sigue siéndolo con la **Cuenta NoCuenta:**
libre de comisiones*, y de lloraditas.

¡Quiero una de esas!

*TIN 0 % y TAE 0 %.

do your thing

Sistemas Concurrentes y Dist...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

a) La secuencia de interfoliación B2 A1 A2 A3 B1 B3 es válida: Esto implica que se ejecutan instrucciones de B antes que las de A, lo cual no es coherente con su definición secuencial.

b) La secuencia de interfoliación B1 A1 A2 B3 A3 B2 es válida: Similar al caso anterior, esta secuencia no sigue el orden secuencial definido para A y B.

c) La secuencia de interfoliación A1 A2 B1 B2 A3 B3 es válida: Esta secuencia respeta el orden secuencial de las instrucciones de A y B, permitiendo que se ejecuten de manera coherente.

Por lo tanto, la opción correcta sería:

c) La secuencia de interfoliación A1 A2 B1 B2 A3 B3 es válida

- Respecto a los semáforos, señalar cual es la opción correcta:

a) Inicialmente, la cola de procesos del semáforo esta vacía

b) En la cola de procesos del semáforo estan todos los procesos que han ejecutado un sem_wait

c) El valor del semáforo puede ser cualquier numero entero

d) La operación sem_wait debe ser atómica, pero no es necesario que lo sea la operación sem_signal

La opción correcta sería:

b) En la cola de procesos del semáforo están todos los procesos que han ejecutado un sem_wait.

Explicación:

a) Inicialmente, la cola de procesos del semáforo está vacía: Esta afirmación no es necesariamente cierta. Dependiendo de la implementación y del momento en que se usen los semáforos, la cola de procesos puede tener procesos en espera desde el inicio si se han realizado operaciones sem_wait antes de que otro proceso intente hacer sem_post.

c) El valor del semáforo puede ser cualquier número entero: Esta afirmación es cierta. El valor de un semáforo puede ser positivo, cero o negativo dependiendo de su implementación y uso. Generalmente, se usa para controlar el acceso a recursos compartidos, pero puede tomar cualquier valor entero.

d) La operación sem_wait debe ser atómica, pero no es necesario que lo sea la operación sem_signal: Ambas operaciones (sem_wait y sem_signal) deben ser atómicas para garantizar la sincronización adecuada entre los procesos. La atomicidad asegura que

la operación se realice completamente sin interrupciones, evitando problemas de concurrencia. Por lo tanto, ambas operaciones suelen ser atómicas en implementaciones de semáforos.

- Supongamos un algoritmo de exclusión mutua (para dos procesos) que cumple todas las propiedades excepto la de espera limitada, siendo ambos procesos bucles infinitos(PE+SC+RS) Entonces:

a) Ambos procesos pueden permanecer en PE indefinidamente

b) Puede ocurrir que un proceso permanezca en PE mientras el otro accede a SC varias veces

c) Un proceso podría quedar en PE indefinidamente mientras el otro esta en RS

d) Los dos procesos podrían estar a la vez en SC

Por lo tanto, la opción correcta sería:

c) Un proceso podría quedar en PE indefinidamente mientras el otro está en RS

Esto significa que un proceso puede quedar atrapado en la sección de entrada (PE) de manera indefinida mientras el otro proceso está en la sección de salida o región crítica (RS). Esta situación es posible cuando no se garantiza la espera limitada, lo que podría ocasionar que un proceso se quede esperando indefinidamente para acceder a la región crítica si el otro proceso no libera el recurso o la sección crítica.

- El siguiente algoritmo de exclusión mutua para 2 procesos:

a) Puede producir interbloqueo

b) No cumple la propiedad de exclusión mutua

c) No cumple la propiedad de espera limitada

d) Todas las afirmaciones anteriores son falsas

La opción correcta es:

a) Puede producir interbloqueo

En este algoritmo, si ambos procesos intentan ingresar a sus respectivas secciones críticas al mismo tiempo, podrían quedar bloqueados. Si el Proceso 0 establece psc[0] en verdadero y luego intenta ingresar a la sección crítica mientras el Proceso 1 tiene psc[1] como verdadero, el Proceso 0 quedará bloqueado en el paso 2 hasta que psc[1]

se establezca como falso, lo que puede no ocurrir si el Proceso 1 también está esperando en su sección de código correspondiente.

Similarmente, si el Proceso 1 intenta ingresar a su sección crítica mientras el Proceso 0 tiene psc[0] como verdadero, el Proceso 1 quedará bloqueado en el paso 2 hasta que psc[0] se establezca como falso, lo que nuevamente puede no suceder si el Proceso 0 está esperando en su sección de código correspondiente.

- Si en el anterior algoritmo permutamos el orden de las líneas 1 y 2:

    a) No cumple la propiedad de progreso en la ejecución

    b) No cumple la propiedad de exclusión mutua

    c) Puede producir interbloqueo

    d) Todas las afirmaciones anteriores son falsas

La permutación no cambia la lógica subyacente del algoritmo, solo altera el orden en que se establecen los semáforos para cada proceso antes de ingresar a la sección crítica.

En este caso, aunque el orden de las líneas 1 y 2 ha cambiado, el algoritmo sigue siendo propenso a interbloqueos. Si ambos procesos intentan ingresar a sus respectivas secciones críticas al mismo tiempo, podrían quedar bloqueados mutuamente, lo que indica que sigue siendo posible el interbloqueo.

Por lo tanto, la respuesta correcta es:

c) Puede producir interbloqueo

- En un monitor con semántica señalar y salir con 2 variables condición declaradas:

    a) El numero total de colas de procesos que gestiona el monitor es 4

    b) El numero total de colas de procesos que gestiona el monitor es 5

    c) El numero total de colas de procesos que gestiona el monitor es 3

    d) El numero total de colas de procesos que gestiona el monitor es 2

Generalmente, se utiliza una cola de espera para cada variable condición, lo que significa que con 2 variables condición declaradas, se necesitarían 2 colas para manejar la espera de los procesos en relación con cada variable condición. Además, hay una tercera cola implícita para los procesos que están fuera del monitor esperando su acceso.

Por lo tanto, la respuesta correcta es:

c) El número total de colas de procesos que gestiona el monitor es 3

- En un monitor ya inicializado, compartido por varios procesos de un mismo programa concurrente:

a) La cola del monitor puede estar vacía o tener máximo un proceso

b) Si hay un proceso dentro del monitor, todos los demás procesos del programa concurrente estarán esperando en la cola del monitor

c) Si la cola del monitor no esta vacía, significa que hay un proceso dentro del monitor

d) Si la cola del monitor esta vacía, significa que hay un proceso dentro del monitor

En un monitor ya inicializado, compartido por varios procesos de un mismo programa concurrente:

c) Si la cola del monitor no está vacía, significa que hay un proceso dentro del monitor.

La cola del monitor puede representar a los procesos que están en espera para acceder a la región crítica o para realizar alguna operación asociada al monitor. Si la cola del monitor no está vacía, indica que hay al menos un proceso esperando su turno para ingresar al monitor o que ya está dentro del monitor ejecutando una sección crítica o realizando alguna operación.

- En un monitor que sigue la semántica Señalar y Salir el proceso que se desbloquea el hacer un signal sobre una variable condición:

a) Se escoge aleatoriamente entre los procesos bloqueados en dicha variable condición

b) Se elige el primer proceso que estuviera esperando para acceder al monitor

c) Ninguna de las respuestas es correcta

d) Se elige el primer proceso que se bloqueó entre los que haya en la cola de dicha variable condición

En un monitor que sigue la semántica Señalar y Salir, al hacer un signal sobre una variable condición:

d) Se elige el primer proceso que se bloqueó entre los que haya en la cola de dicha variable condición.

Al utilizar la operación signal en una variable condición en un monitor, se libera uno de los procesos que estén esperando en esa variable condición. En este contexto, generalmente se selecciona el primer proceso que se bloqueó y está en espera dentro de la cola de esa variable condición, permitiéndole acceder al monitor. Esto sigue el principio de justicia básica, donde el primero en bloquearse es el primero en recibir la señal para desbloquearse y acceder al recurso protegido por el monitor.

2. Un proceso productor y tres procesos consumidores (Consumidor_i, i = 0, 1 , 2) se sincronizan de tal forma que cada dato producido por el productor debe ser consumido por los tres consumidores. El productor no podrá producir un nuevo dato hasta que los tres consumidores hayan consumido el anterior y los consumidores no podrán consumir hasta que el productor produzca un nuevo dato. Se supone que dato es una variable compartida por los 4 procesos que tiene tipo entero. Asegurar la sincronización requerida por los procesos mediante el uso de semáforos.

{Declaración e inicialización de variables globales y semáforos}

```
int dato;                  //Variable compartida
Semaphore sem_pro = 1;     // Semáforo para el productor
Semaphore sem_con = 0;     //Semáforo para el consumidor
Semaphore mutex = 1;       //Semáforo mutex para garantizar exclusión mutua
```

Productor

```
While(true)
    begin
    Produce(dato);         //Produce un nuevo dato
    wait(sem_con);         //Espera a que los consumidores hayan consumido el dato
    signal(sem_pro);       //Permite al productor producir un nuevo dato
    end
```

Consumidor(i) i =0,1,2

```
While(true)
    begin
    wait(sem_pro);         //Espera a que el productor produzca un nuevo dato
    wait(mutex);           //Entra en la sección crítica
    Consume(dato);         //Consume el dato producido por el productor
```

signal(mutex);//Sale de la sección critica

signal(sem_con);//Indica al productor que el dato ha sido consumido

end

3. Dado el siguiente fragmento de programa concurrente P (programa de la izquierda), donde se consideran a las sentencias encerradas entre <…> como atómicas.

3.1 Construir el grafo de precedencia asociado a P dibújalo justo a la derecha del programa P.

int x,y,z;

Begin

x:= 3; y:= 2 ; z:= 1;

{P0}

cobegin

begin

<x:= x+1; > **{P1}**

<z:= z-y; > **{P2}**

<y:= x -1 > **{P3}**

end

<x:= y+z; > **{P4}**

coend

<print(x,y); > **{P5}**

End

El grafo de precedencia es una representación visual que muestra la relación entre las diferentes tareas o procesos en un sistema concurrente, indicando la dependencia entre ellos. Para el programa dado, se pueden identificar ciertas dependencias entre las operaciones para construir el grafo de precedencia.

Aquí hay una descripción de las dependencias:

1. x:= 3; y:= 2 ; z:= 1; (Inicio)
2. <x:= x+1; > {P1}: Depende de la inicialización de x.
3. <z:= z-y; > {P2}: Depende de la inicialización de z y y.
4. <y:= x -1 > {P3}: Depende de la inicialización de x.
5. <x:= y+z; > {P4}: Depende de las operaciones de P1, P2 y P3.
6. <print(x,y); > {P5}: Depende de la operación P4.

Ahora, construyamos el grafo de precedencia:

x:= 3; y:= 2 ; z:= 1; (Inicio)

|

|

V

P1 (x:= x+1;)

|

|

V

P3 (y:= x -1)

**/ **

**/ **

V V

P2 P4

(z:= z-y) (x:= y+z;)

\ /

\ /

V V

P5 (print(x,y);) (Final)

El grafo de precedencia muestra la secuencia en la que las operaciones deben completarse para garantizar que todas las dependencias se respeten. Por ejemplo, P1 depende de la inicialización de x, P3 depende de P1, P2 y la inicialización de y y z, y así sucesivamente. Esto refleja la relación de dependencia entre las diferentes operaciones y la necesidad de que se completen en un orden específico para garantizar la consistencia de los datos y el resultado final correcto.

3.2 De los fragmentos de programas que aparecen a la derecha, indica si son o no equivalentes al fragmento P. Para ello, construir su grafo de precedencia justo debajo y compararlo con el obtenido en el apartado 3.1

a)

Begin

P0;

P1;

fork P4;

fork P2;

P3;

join P2;

join P4;

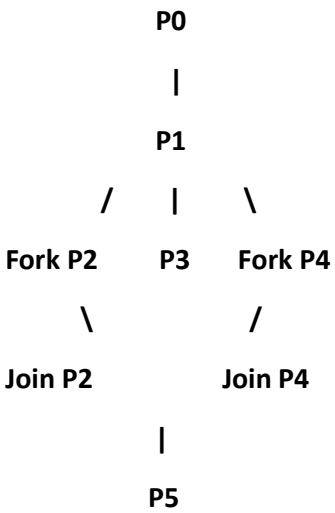
P5;

End

El grafo de precedencia mostrará las dependencias entre estas operaciones:

1. P0;: Operación inicial.
2. P1;: Depende de la finalización de P0.
3. fork P4;: Crea una bifurcación, permite que P4 se ejecute en paralelo con las operaciones restantes.
4. fork P2;: Crea otra bifurcación, permite que P2 se ejecute en paralelo con las operaciones restantes.
5. P3;: Es una operación secuencial que depende de la finalización de P1.
6. join P2;: Espera a que P2 finalice antes de continuar con las operaciones siguientes.
7. join P4;: Espera a que P4 finalice antes de continuar con las operaciones siguientes.
8. P5;: Operación final que depende de la finalización de P3, P2 y P4 (ya que los joins deben esperar a que sus respectivos forks finalicen).

El grafo de precedencia se ve de la siguiente manera:



b)

Begin

P0;

fork P4;

P1;

P2;

P3;

join P4;

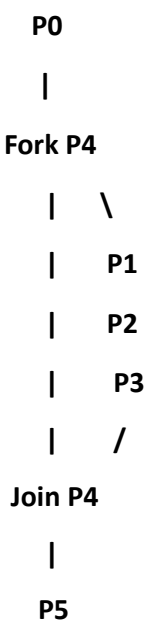
P5;

End

Vamos a representar las dependencias entre estas operaciones:

1. P0;: Operación inicial.
2. fork P4;: Crea una bifurcación, permitiendo que P4 se ejecute en paralelo con las operaciones restantes.
3. P1; P2; P3;: Son operaciones secuenciales que pueden ejecutarse independientemente.
4. join P4;: Espera a que P4 finalice antes de continuar con las operaciones siguientes.
5. P5;: Operación final que depende de la finalización de P4 (ya que el join espera a que P4 finalice).

El grafo de precedencia se vería así:



3.3 Indicar todas las posibles interfolaciones de sentencias atómicas que pueden derivarse de la ejecución concurrente de P así como los valores que se imprimen (x e y) en cada caso. Hazlo rellenando las filas necesarias de la siguiente tabla.

1. P0 -> P1 -> P2 -> P3 -> P4 -> P5 (x = 5 , y = 4)

2. P0 -> P4 -> P1 -> P2 -> P3 -> P5 (x = 6 , y = 5)

3. P0 -> P1 -> P2 -> P3 -> P5 -> P4 (x = 5 , y = 4)

5. La conocida atracción "Splash" de un parque acuático tiene un aforo limitado a 30 usuarios, que disfrutan de la atracción durante un tiempo que depende del usuario y después salen por la puerta de la salida. No obstante, dicha atracción necesita un mantenimiento periódico reglamentario realizado por el especialista cada vez que la usan 200 usuarios. Este mantenimiento requiere que la atracción no este siendo usada (este vacía), y después del mantenimiento, podrán seguir entrando usuarios a la atracción hasta el siguiente ciclo de mantenimiento. De esta manera, cuando el usuario 200 desde el ultimo ciclo de mantenimiento, no se dejara entrar a otros usuarios y, cuando la atracción se vacié por completo, podrá actuar el especialista de mantenimiento. Una vez el especialista finaliza su trabajo, deberá esperar hasta que otros 200 usuarios hayan pasado por la atracción(y hayan salido de la misma) para poder volver a hacer el mantenimiento. Diseñar el monitor Splash, con semántica SU, que resuelva la sincronización requerida para el problema, teniendo en cuenta que dicho monitor se usará por parte de los procesos usuario y especialista de acuerdo al siguiente esquema:

Proceso Usuario (i), i = 1,…,N

```
begin
…
Splash.entrada();
{Disfrute de la atracción}
Splash.salida();
…
end
```

Proceso Especialista

```
begin
while true do begin
        Splash.espera();
        {Trabajos de mantenimiento}
        Splash.fin_trabajo();
        end
end
```

MONITOR SPLASH

Monitor Splash

```
  var
    usersWaiting: int = 0
    usersInside: int = 0
    maintenanceCycle: int = 0
    userCountSinceLastMaintenance: int = 0

  condition notFull, notEmpty, maintenance, finishedMaintenance

  procedure entrada()
    if userCountSinceLastMaintenance == 200 then
      wait(maintenance)
      userCountSinceLastMaintenance = 0
      signal(finishedMaintenance)

    if usersInside >= 30 or usersWaiting > 0 then
      usersWaiting++
      wait(notFull)
```

```
        usersWaiting--

    usersInside++
    userCountSinceLastMaintenance++
    signal(notEmpty)

procedure salida()
    usersInside--
    signal(notFull)

procedure espera()
    if usersInside > 0 then
        wait(notEmpty)

procedure fin_trabajo()
    signal(maintenance)
    wait(finishedMaintenance)
```