Preguntas-de-conceptos-Resueltas...

pr0gramming_312823

Sistemas Concurrentes y Distribuidos

2º Grado en Ingeniería Informática

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

Test de conceptos SCD T1-T4

(T1) 1. Indicar cuál de las siguientes definiciones se aplica al concepto de programa concurrente correcto (las demás definiciones no consiguen explicar este concepto de una forma totalmente satisfactoria)

a. Si se puede demostrar que el programa cumple las propiedades de seguridad y vivacidad de los procesos.
b. Si todos los procesos acceden a recursos compartidos con *justicia* y se obtiene un resultado secuencial correcto.
c. Si se puede demostrar que todos los procesos del programa cumplen todas las propiedades concurrentes siguientes: *seguridad, vivacidad y equidad*
d. Si podemos demostrar que todos los procesos terminan sus cálculos y, por tanto, es equivalente a una determinada secuencia de ejecución secuencial

(T1) 2. Seleccionar la única respuesta correcta. Un programa concurrente que cumpla la propiedad de seguridad para todas sus ejecuciones se considerará correcto si además:

a. Todos sus procesos también cumplen la propiedad de *vivacidad*, lo que es equivalente a afirmar que los procesos del programa nunca pueden llegar a una situación de interbloqueo
b. Se puede demostrar que sus procesos no sufren inanición en ninguna posible ejecución del programa
c. Sus procesos consiguen ejecutar sus instrucciones de forma equitativa
d. Se puede demostrar que sus procesos no sufren inanición en ninguna posible ejecución del programa y además se ha de cumplir la propiedad de vivacidad

(T1) 3. Seleccionar la única respuesta correcta sobre el concepto de *condición de carrera* de los programas concurrentes:

a. Se refiere a que no podemos saber cuál de los procesos del protocolo para resolver el problema de la exclusión mutua conseguirá entrar primero en sección crítica
b. Cuando se produce no se cumple la propiedad de vivacidad de algunos procesos del programa concurrente
c. Múltiples procesos consiguen llegar a cumplir esta condición si el resultado de sus cálculos depende del orden en que se ejecuten
d. Esta condición se produce cuando se programan bucles de espera activa y no se puede saber cuál de los procesos que llegan al bucle conseguirá terminarlo primero

(T1) 4. Indicar cuál de los siguientes es la que mejor se adapta al concepto de *vivacidad* (ausencia de *inanición*) de un proceso que se ejecuta como parte de un programa concurrente:

a. Nunca se puede bloquear durante su ejecución
b. Si el código del proceso contiene instrucciones condicionales, hemos de poder asegurar que dichas instrucciones se ejecutarán frecuentemente si las condiciones de las que depende consiguen un valor de verdad favorable muy a menudo.
c. El proceso no puede sufrir un retraso indefinido en la ejecución de sus instrucciones, aunque no podemos dar una indicación precisa de cuándo terminará dicho retraso
d. Esta condición se produce cuando se programan bucles de espera activa y no se puede saber cuál de los procesos que llegan al bucle conseguirá terminarlo primero

(T1) 5. ¿En qué consiste la ventaja de la programación concurrente si sólo tenemos un procesador en nuestro computador? (justificar la respuesta elegida):

a. Desacopla la ejecución de los procesos dedicados a entradas/salidas
b. Se puede crear una hebra de ejecución independiente para realizar los cálculos de cada petición que se reciba de un programa cliente que se ejecuta en otro procesador
c. El programa tardará menos en ejecutarse porque ahora no se para nunca en las operaciones de entrada/salida
d. Los compiladores de los lenguajes con procesos concurrentes generan código optimizado

(T1) 6. La verificación utilizando la lógica de programas (pre, post-condiciones e invariantes) no se puede aplicar directamente a la verificación de los programas concurrentes por una de las razones siguientes(justificar la respuesta elegida):

a. La lógica de programas sólo tiene aplicación a la demostración de corrección de un algoritmo secuencial porque, a diferencia de los programas concurrentes, siempre termina y producen unos resultados
b. Porque falta una regla de verificación para componer las demostraciones secuenciales de los procesos del programa concurrente, que realizamos utilizando la lógica de programas
c. Porque aplicando la lógica de programas para verificar las instrucciones de los procesos de un programa concurrente no se pueden demostrar propiedades de vivacidad de dichos procesos
d. Puede existir interferencia entre los predicados (pre y post-condiciones) con que anotamos las instrucciones que programamos en cada proceso concurrente del programa y la ejecución de instrucciones atómicas de los otros procesos

(T1) 7. En el modelo abstracto de la programación concurrente se incluye la condición de "progreso finito" de los procesos de un programa concurrente. Seleccionar cuál de las siguientes afirmaciones sobre dicha hipótesis es cierta.(justificar la respuesta elegida):

a. Los procesos de un programa concurrente nunca entrarán en bucles indefinidos de espera activa
b. Los procesos de un programa concurrente no se pueden suspender o bloquear en ningún caso
c. Los procesos de un programa concurrente siempre tienden a ejecutar las instrucciones de su código, pero esto no nos asegura que siempre consigan avanzar en los cálculos que han de realizar
d. Es equivalente a afirmar que los procesos de un programa concurrente nuca puedan sufrir inanición

(T2) 1. Seleccionar la única respuesta correcta de las siguientes.

a. Un programa con procesos concurrentes y programado con monitores no puede implementarse utilizando procesadores multinúcleo pues estos poseen caches privados
b. La protección automática de las variables permanentes declaradas dentro de un monitor hace innecesaria la demostración de la propiedad concurrente denominada seguridad de los programas concurrentes
c. La planificación FIFO obligatoria de las colas de los monitores asegura en todo caso que siempre se satisfaga la propiedad de vivacidad de los procesos de un programa concurrente
d. Como consecuencia de haber realizado una llamada a un procedimiento de un monitor, un proceso concurrente se puede suspender y salir del monitor

(T2) 2. La interpretación correcta del axioma de la operación de sincronización "c.signal()" con señales desplazantes {not empty(c) and L and C} c.signal() {IM and L } es una de las siguientes (Nota: IM= invariante de monitor, L= invariante de las variables locales al procedimiento, C= condición de sincronización para la variable condición implicada).

a. El proceso concurrente que ocasiona la ejecución de la operación de sincronización no se suspende salvo que la cola "c" esté vacía
b. El proceso concurrente se suspende en la cola de entrada al monitor cuando ejecuta la operación de sincronización
c. Cuando se ejecuta la operación de sincronización dentro de un procedimiento del monitor, el estado reentrante del procedimiento hace cierta la condición "C"
d. Si la cola no está vacía, el valor de certeza de la condición "C" se transmite a todo el monitor hasta que se desbloquea 1 proceso de la cola

(T2) 3. La interpretación correcta del axioma de la operación de sincronización c.wait() con señales desplazantes ({ IM and L } c.wait() { C and L }) es una de las siguientes.

a. Si no se cumple la condición "C", al volver la llamada a la operación de sincronización, el procedimiento del monitor no se ha programado correctamente
b. El proceso que ocasiona la ejecución de la operación se bloquea y deja libre el monitor, cumpliéndose a continuación la condición "C" como precondición de la siguiente instrucción
c. La condición "C" coincide con el invariante global del monitor
d. Si no se cumple la condición "C", al volver la llamada a la operación de sincronización, no podemos afirmar que el invariante del monitor se cumpla

(T2) 4. Seleccionar la única respuesta correcta:

a. Monitores con métodos largos (mucho código) ocasionan que la ejecución de un programaconcurrente sea prácticamente secuencial, ya que sólo un proceso concurrente puede entrar al monitor cada vez
b. Un proceso de un programa concurrente podría llegar a bloquearse antes de que termine la ejecución del método del monitor al que previamente ha llamado
c. Los métodos de los monitores no admiten paso de parámetros por referencia por que se podrían pasar variables globales del programa como argumentos
d. A diferencia de lo que ocurre con las clases en los lenguajes de programación orientados aobjetos, los monitores no se pueden instanciar ya que el invariante ha de ser único

(T2) 5. Seleccionar la única respuesta correcta:

a. Un programa con procesos concurrentes y programado con monitores no puede implementarse en procesadores multinúcleo- sólo se puede en monoprocesadores con memoria globalmente consistente
b. Como consecuencia de la llamada al procedimiento de un monitor, un proceso de un programa concurrente se puede suspender y salir del monitor
c. La protección automática de las variables permanentes declaradas dentro de un monitor hace innecesaria la demostración de la propiedad de seguridad en los programas
d. La planificación FIFO obligatoria de las colas de los monitores asegura en todos los casos que se satisface la propiedad de vivacidad de los procesos de un programa concurrente

(T2) 6. Aunque un monitor garantiza el acceso en exclusión mutua a sus variables permanentes, los métodos de una clase-monitor han de cumplir la propiedad denominada reentrancia (un procedimiento puede ser interrumpido en medio de su ejecución y volver a ser llamado después por otros procesos del programa de manera segura). Seleccionar una de las razones siguientes para que un lenguaje con monitores exija la propiedad de reentrancia:

a. Porque en los métodos de cada monitor se pueden programar varias operaciones c.wait()
b. Porque durante la ejecución de un programa, los procesos concurrentes van alternándose enla entrada al monitor
c. Sólo porque un determinado proceso podría entrar varias veces en un mismo monitor y tener aún pendiente la terminación de alguna de sus llamadas anteriores
d. Porque se pueden anidar las llamadas a los procedimientos de un mismo monitor y un proceso podría tener pendientes de terminación más de 1 llamada

(T2) 7. Sobre el significado de la semántica de las operaciones de variables condición de los monitores, seleccionar la única respuesta correcta:

a. Las "señales automáticas" se consideran señales desplazantes pero no hay que demostrar la corrección de las operaciones "c.signal()"
b. Con la semántica denominada "señalar y continuar" (SC) se puede producir "robo de señal" (un tercer proceso entra en el monitor antes que el proceso señalado y cambia el valor de la condición de desbloqueo) pero sólo si todos los procesos se bloquean en la misma cola
c. Con semántica de "señales urgentes" puede ocurrir que la salida de un proceso del monitor provoque que un proceso suspendido en una cola del monitor vuelva a ejecutarse
d. Con semántica de señales "señalar y salir" (SX) se obliga a que las operaciones c.signal() se programen necesariamente como la última instrucción de los procedimientos del monitor

(T2) 8. 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. Un proceso podría quedar en PE indefinidamente mientras el otro está en RS
c. Puede ocurrir que un proceso permanezca en PE mientras el otro accede a SC varias veces
d. Todas las anteriores son falsas

(T2) 9. En un programa concurrente con semáforos, en un momento determinado hay dos procesos bloqueados en el semáforo *sem* y otro proceso hace un sem_signal(*sem*). El efecto de dicho sem_signal es:

a. Liberar a uno de los procesos bloqueados en *sem*
b. Incrementar el valor del semáforo en una unidad
c. Liberar a los dos procesos bloqueados en *sem*
d. Liberar a un proceso bloqueado en *sem* e incrementar el valor del semáforo en una unidad

(T2) 10. En un monitor ya inicializado y que está siendo utilizado por varios procesos:

a. La cola del monitor puede estar vacía o tener como máximo un proceso
b. Si hay un proceso dentro del monitor, todos los demás procesos estarán esperando en la cola del monitor
c. Si la cola del monitor está vacía, significa que hay un proceso dentro del monitor
d. Todas las anteriores son falsas.

(T2) 11. En un programa concurrente que utiliza el semáforo *sem*:

a. Si hay algún proceso en la cola del semáforo *sem*, el valor de dicho semáforo es cero en ese momento
b. Por la cola de procesos del semáforo *sem* pasan todos los procesos que han ejecutado un sem_wait(*sem*)
c. Un proceso no puede ejecutar dos operaciones sem_wait seguidas sobre el semáforo *sem*
d. Todas las anteriores son verdaderas.

(T2) 12. En un monitor con una semántica de señales "señalar y esperar" (SE) y sobre el que se han declarado 2 variables de tipo condición:

a. El número total de colas de procesos que gestiona el monitor es 4
b. El número total de colas de procesos que gestiona el monitor es 5
c. El número total de colas de procesos que gestiona el monitor es 3
d. El número total de colas de procesos que gestiona el monitor es 2

(T2) 13. Supongamos un algoritmo de exclusión mutua para dos procesos 1 y 2, que cumple la propiedad de exclusión mutua y que está diseñado de forma tal que, cuando uno de ellos entre a SC, el otro (aunque esté en RS) tiene garantizado que accederá a SC antes de que el primero vuelva a entrar en la SC:

a. El algoritmo no cumple la propiedad de *espera limitada*
b. El algoritmo no cumple la propiedad de progreso en la ejecución
c. El algoritmo puede producir interbloqueo
d. El algoritmo es correcto

(T2) 14. En un programa concurrente que utiliza el semáforo *sem*, que está inicializado a cero:

a. La primera operación sem_wait sobre el semáforo *sem* bloqueará siempre al proceso que la ejecuta
b. La primera operación sem_signal sobre el semáforo *sem* no tiene ningún efecto
c. No se puede hacer un sem_wait(*sem*) hasta que no se haga un sem_signal(*sem*)
d. Todas las anteriores son falsas

(T2) 15. Respecto al concepto de monitor:

a. Las variables permanentes pueden ser accedidas fuera del monitor, siempre que no se modifique su valor
b. Si un procedimiento está ejecutando un procedimiento del monitor (A), ningún otro proceso podrá ejecutar el procedimiento A pero sí otro procedimiento de dicho monitor
c. Si un procedimiento está ejecutando un procedimiento del monitor (A), ningún otro proceso podrá ejecutar el procedimiento A ni ningún otro procedimiento de dicho monitor
d. La a) y la b) son correctas

(T2) 16. Respecto a los cerrojos, señalar la opción correcta:

a. Los cerrojos son un mecanismo que no usa espera ocupada
b. Los cerrojos utilizan instrucciones máquina atómicas
c. La instrucción "LeerAsignar(a)" utiliza una variable booleana
d. La b) y la c) son correctas

(T2) 17. Respecto a un programa concurrente que utiliza un monitor:

a. Todas las hebras del programa deben llamar a algún procedimiento del monitor
b. Lo único que las hebras del programa pueden hacer sobre el monitor es invocar a los procedimientos exportados de dicho monitor
c. El monitor es un proceso que se ejecuta concurrentemente con el resto de hebras del programa
d. Todas las anteriores son falsas

(T3 y T4) 1. Seleccionar la alternativa verdadera:

a. En ningún caso el proceso receptor se suspenderá al ejecutar la orden receive(…) en el paso *de mensajes asíncrono con búfer*
b. Las condiciones de la instrucción de espera selectiva (select) han de ser excluyentes entre las distintas alternativas de esta orden
c. Una orden select podría suspender su ejecución incluso si todas las condiciones de sus alternativas se hubieran evaluado como ciertas
d. El paso de mensajes síncrono no puede implementarse con un búfer

(T3 y T4) 2. Seleccionar la alternativa verdadera:

a. El algoritmo RMS nos dice que la planificabilidad (las tareas consiguen ejecutar sus unidades antes de expire su plazo máximo de respuesta) de un conjunto de N tareas periódicas es imposible si la utilización conjunta del procesador supera el límite: $U=N*(2\text{^}(1/N)-1)$
b. Si se utiliza asignación estática de prioridades a las tareas, entonces podemos afirmar que dicho conjunto de tareas es planificable independientemente de la fase de cada tarea
c. El plazo de respuesta máximo (D) para cada tarea no depende del instante de su próxima activación
d. El algoritmo de planificación denominado RMS nunca puede proporcionarnos un esquema de planificación de tareas con aprovechamiento del 100% del tiempo del procesador

(T3 y T4) 3. Seleccionar las afirmaciones correctas respecto de la ejecución de las alternativas de la espera selectiva **select** que poseen algunos lenguajes de programación distribuida:

a. Si hay guardas ejecutables en las alternativas, se selecciona no determinísticamente una entre las que poseen una orden **send** ya iniciada
b. Si no hay guardas ejecutables pero sí las hay potencialmente ejecutables, la instrucción de espera selectiva se suspende hasta que un proceso nombrado inicie una operación send
c. Si en las alternativas no se ha programado ninguna sentencia de entrada (**receive**), se selecciona no determinísticamente una cualquiera de éstas para su ejecución y la espera selectiva termina
d. La espera selectiva nunca puede levantar una excepción en el programa donde se programe

(T3 y T4) 4. Seleccionar la afirmación correcta respecto de la espera selectiva con guardas indexadas

a. Las condiciones de las alternativas de la espera selectiva no pueden depender de argumentos de entrada (arg) de las sentencias de entrada (receive (var arg))que se programen en éstas
b. El índice que se programa para replicar una alternativa no puede depender de valores límite (inicial, final) no conocidos en tiempo de compilación del programa
c. En la instrucción de espera selectiva no se pueden combinar alternativas indexadas con otras alternativas normales no indexadas
d. Un conjunto de N procesos emisores cada uno envía caracteres al resto de los procesos de dicho conjunto, es decir, cada proceso recibe (N-1) mensajes de los demás, entonces es imposible programar los procesos individuales con instrucciones de espera selectiva porque dicha orden se ejecuta 1 vez y termina

(T3 y T4) 5. Seleccionar la afirmación correcta

a. La instrucción de espera selectiva no es necesaria en los lenguajes de programación porque todo se puede programar con operaciones de paso de mensajes no bloqueantes
b. **MPI_Probe** o comprobación bloqueante de mensaje es redundante con **MPI_Wait**
c. MPI_Send (*con soporte hardware*) podría volver sin esperar la ejecución de la operación de recepción concordante
d. **MPI_Recv** podría volver sin esperar la ejecución de la operación de envío concordante

(T3 y T4) 6. Seleccionar la única respuesta correcta respecto de la orden MPI_Receive:

a. El proceso que programe dicha orden siempre se bloquea independientemente del estado del búfer (vacío o con datos esperando)
b. Sólo si suponemos un mecanismo de comunicación síncrono sin búfer ("citas") un proceso que programe dicha orden siempre se bloqueará
c. Si la ejecución de la orden anterior no bloquea al proceso, entonces no podemos asegurar que la operación de transmisión de los datos sea segura
d. El proceso que programe dicha orden se bloqueará sólo si el búfer es encuentra vacío, es decir, no hay mensajes pendientes de ser recibidos

(T3 y T4) 7. Seleccionar la única respuesta correcta en el caso de que un proceso programe la operación insegura MPI_Irecv(…):

a. La reducción del tiempo de recepción del mensaje en el receptor es independiente de que el sistema de ejecución cuente con hardware especializado o no
b. Los datos ya transmitidos siempre se mantienen el búfer del sistema hasta que el proceso receptor pueda descargarlos a su espacio de memoria y esto es independiente de que el proceso que ejecute la orden vuelva inmediatamente (sistema con hardware especializado)
c. Con esta operación no se iniciará la transmisión de datos entre el proceso emisor y el receptor inmediatamente
d. Con esta operación siempre se anulará el tiempo de espera en el proceso receptor

(T3 y T4) 8. La elección y ejecución inmediata de 1 alternativa de la orden de espera selectiva (*select*) sólo se producirá si se cumple una de las condiciones siguientes (indicar cuál):

a. Sólo depende de que exista alguna orden potencialmente ejecutable en ese momento
b. Sólo depende de que alguna condición de las órdenes con guarda sea cierta
c. Existe, al menos, una orden potencialmente ejecutable y además se nombra a un proceso del programa que ya ha iniciado su envío
d. Sólo de que exista algún proceso del programa que haya iniciado su envío

(T3 y T4) 9. Indicar cuál de las siguientes afirmaciones puede considerarse correcta respecto de la orden de espera selectiva:

a. Una vez que se ha seleccionado una orden potencialmente ejecutable, el proceso que la contiene se ejecuta secuencialmente hasta que termina el bloque componente (*do…end*)
b. Una orden de espera selectiva no termina hasta que no ejecute cada una de sus alternativas, lo cual ocurre no determinísticamente
c. Una orden de espera selectiva puede entremezclar las secuencias de ejecución de las instrucciones de los bloques de sentencias componentes de cada una de sus alternativas
d. La orden de espera selectiva puede programar un array de guardas indexadas, pero cada una ha de incluir una operación de entrada (*receive(…)*)

(T3 y T4) 10. Indicar cuál de las siguientes afirmaciones es correcta con respecto a la denominada deriva acumulativa que experimentan los programas de tiempo real:

a. La deriva acumulativa se produce en una tarea porque no es posible programar un retraso exacto hasta un instante en que produzca la siguiente activación, es decir, todos los retrasos que se programan indican un intervalo de suspensión relativo al instante en que se ejecutan
b. Se puede eliminar la deriva de los retrasos relativos programados en cada tarea pero nunca se podrán eliminar completamente los retrasos causados por el sistema operativo
c. La orden **sleep_until(…)** permite eliminar completamente la deriva acumulativa
d. Siempre es posible programar una tarea periódica que se active transcurrido un periodo de tiempo exacto desde su última activación en todas sus ejecuciones

(T3 y T4) 11. Respecto del esquema de planificación dinámica de tareas EDF, indicar cuál de las siguientes afirmaciones es la correcta

a. La aplicación de este método es incompatible con aplicar el RMS
b. Un conjunto de tareas cuya utilización del procesador es del 100% y que resulta ser planificable con el algoritmo EDF podría ser también planificable con RMS
c. EDF es el único esquema de planificación dinámico
d. Siempre podremos planificar con EDF cualquier conjunto de tareas cuya utilización del procesador no sea mayor del 100%, independiente de valor de su plazo de tiempo límite

Cuestiones y Preguntas

(T1) **Explicar el significado de las propiedades de corrección de los programas concurrentes y cuándo podemos afirmar que un programa concurrente es correcto y en qué grado de corrección estaría**

- Propiedad de "**Seguridad**": se refiere a condiciones que deben cumplirse en cada instante, del tipo: "nunca pasará nada malo". Ejemplos de propiedades de este tipo son: "Exclusión mutua"(2 procesos nunca entrelazan ciertas subsecuencias de operaciones); "Ausencia de Interbloqueo" (nunca ocurrirá que los procesos se encuentren esperándose unos a otros por algo que jamás sucederá); "Productor-Consumidor Seguro": el consumidor debe consumir todos los datos producidos por el productor en el orden en que se van produciendo, etc.
- Propiedad de "**Vivacidad**": se refiere propiedades que deben cumplirse eventualmente, del tipo: "realmente sucede algo bueno". Un ejemplo de propiedad de este tipo es: "Ausencia de inanición" de los procesos de un programa concurrente (un proceso o grupo de procesos no puede ser indefinidamente pospuesto, en algún momento, podrá avanzar).
- Propiedad secundaria "**Equidad**" en la ejecución de los procesos: un proceso que desee progresar debe hacerlo con justicia relativa con respecto a los demás. Más ligado a la implementación y a veces incumplida por los programas concurrentes: existen distintos grados de cumplimiento de esta propiedad, que determinan diferentes sub-propiedades.

Un programa concurrente para ser considerado correcto ha de cumplir obligatoriamente las propiedades de "Seguridad" y "Vivacidad". Si además garantiza la "Equidad" de ejecución de los procesos podríamos decir que es completamente correcto.

(T2) Programar el mecanismo de sincronización denominado *barrera* con monitores. Se necesita programar un procedimiento del monitor para implementar la espera de los procesos que esperan y otro para desbloquear a todos, que invoca el último proceso esperado

```
Monitor Barrera;
Const
 M:integer;
//forman el grupo en la
//barrera

Var
 n: integer;
 s: condition;

procedure
entrada_salida();
```

```
begin
  n:= n + 1;
  if (n < M) then
     s.wait()
else
 begin
   n:= 0;
   s.signal_all();
 end;
end;
begin
   n:= 0;
end;
```

(T3 y T4) Explicar el problema (seguridad) que tendría el siguiente código, que programa 2 operaciones de paso de mensajes no bloqueante y sin búfer con respecto a los valores finales de las variables x e y, que aparecen señaladas con (**) el código mostrado:

```
int main(int argc, char*argv[]){
int rank, size, vecino, x, y;
MPI_Status status;
MPI_request request_send, request_recv;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
y=rank*(rank+1);
if (rank%2) vecino= rank+1;
else vecino= rank – 1;

//Para rellenar


//Las siguientes operaciones pueden aparecer en cualquier orden
MPI_Irecv(&x,1,MPI_INT,vecino,0,MPI_COMM_WORLD,&request_recv);
MPI_Isend(&y,1,MPI_INT,vecino,0,MPI_COMM_WORLD,&request_send);

MPI_Wait(&request_send, &status);
MPI_Wait(&request_recv, &status);

//(**)
x= y + 7;
```

El código inicial anterior produciría un error de ejecución puesto que las operaciones MPI_Isend(…) y MPI_Irecv(…) son no-bloqueantes, es decir, devuelven el control al programa antes incluso de que sea seguro modificar los datos. Consecuentemente, el código señalado (**) no puede utilizar la variable “y” ni alterar de ninguna manera el valor de la variable “x”.