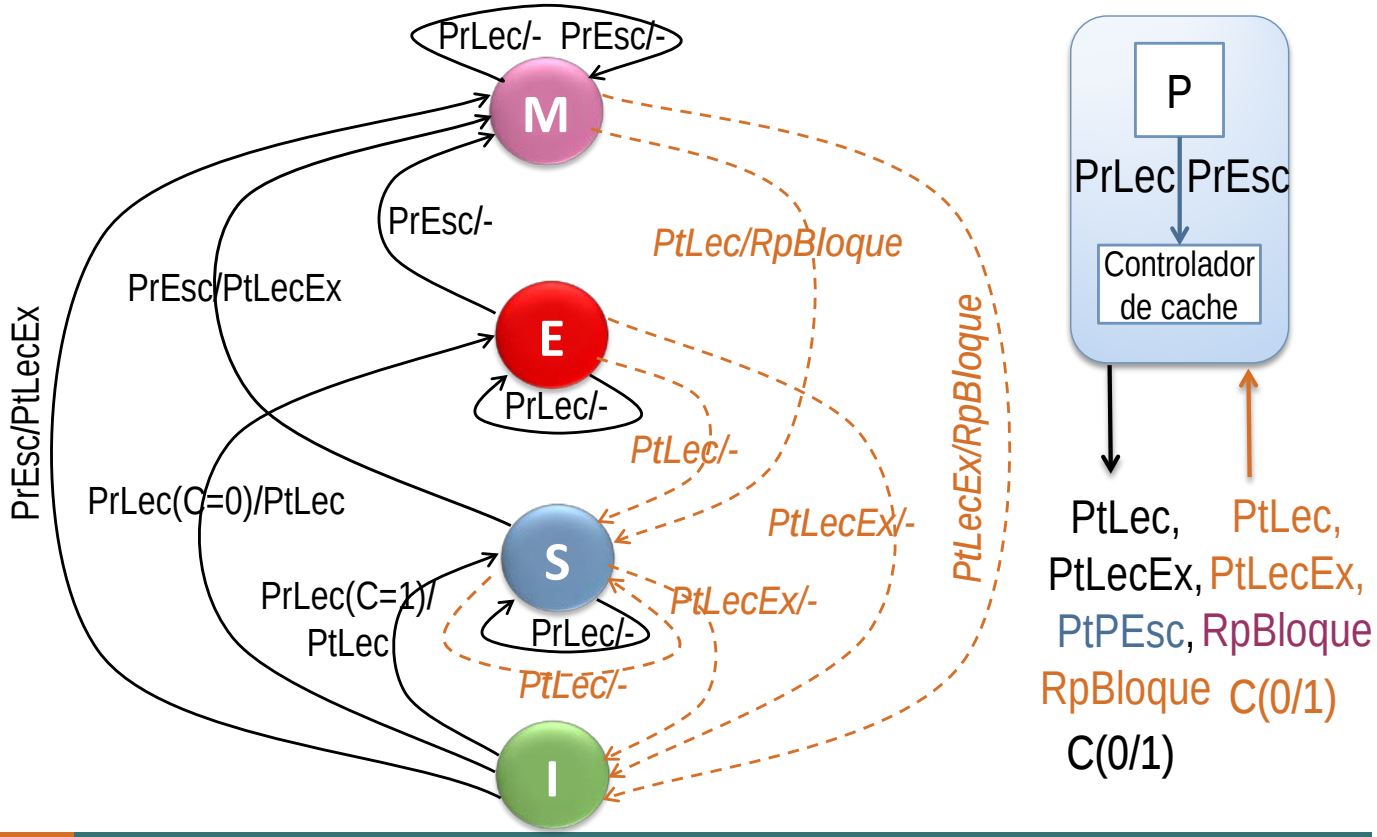
Ejercicio 1. En un multiprocesador SMP con 4 procesadores o nodos (N0-N3) basado en un bus, que implementa el protocolo MESI para mantener la coherencia, supongamos una dirección de memoria incluida en un bloque que no se encuentra en ninguna cache. Indique los estados de este bloque en las caches y las acciones que se producen en el sistema ante la siguiente secuencia de eventos para dicha dirección:

1. Lectura generada por el procesador 1
2. Lectura generada por el procesador 2
3. Escritura generada por el procesador 1
4. Escritura generada por el procesador 2
5. Escritura generada por el procesador 3

Diagrama MESI de transiciones de estados



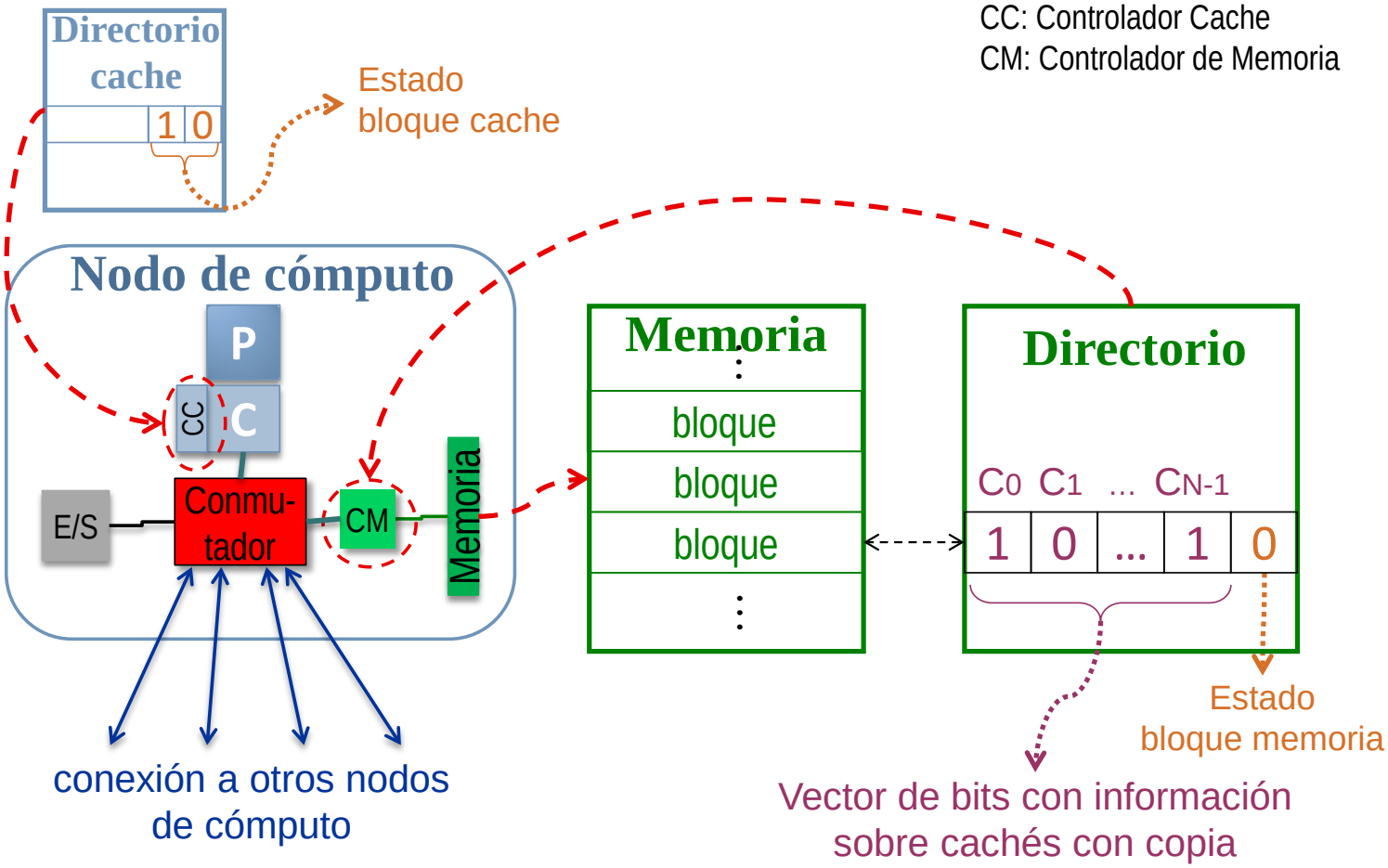
Datos del ejercicio

Se accede a una dirección de memoria cuyo bloque no se encuentra en ninguna cache, luego debe estar actualizado en memoria principal.

Hay 4 nodos con cache y procesador (N0-N3). Intervienen N1, N2 y N3.

ESTADO INICIAL	EVENTO	ACCIÓN	ESTADO SIGUIENTE
N1) Inválido **N2) Inválido** **N3) Inválido**	P1 lee k	- N1 genera petición de lectura del bloque k (PtLec(k)) - N1 recoge la respuesta con el bloque (RpBloque(k)) depositada en el bus por la memoria principal, el bloque entra en la cache de N1 en estado exclusivo ya que no hay copia en otra cache del bloque (la salida de la OR cableada será 0).	**N1) Exclusivo** **N2) Inválido** **N3) Inválido**
N1) Exclusivo **N2) Inválido** **N3) Inválido**	P2 lee k	- N2 genera PtLec(k) - N1 observa PtLec(k) y, como tiene el bloque en estado exclusivo, lo pasa a compartido (la copia que tiene ya no es la única en caches) - N2 recoge RpBloque(k) que ha depositado la memoria, el bloque entra en estado compartido en la cache de N2.	**N1) Compartido** **N2) Compartido** **N3) Inválido**
N1) Compartido **N2) Compartido** **N3) Inválido**	P1 escribe en k	- N1 genera petición de lectura con acceso exclusivo del bloque k (PtLecEx(k)) (suponemos que no hay petición de acceso exclusivo sin lectura) - N2 observa PtLecEx(k) y, como tiene el bloque en estado compartido, lo invalida. - N1 no recoge RpBloque(k) depositada por la memoria (la petición también es de lectura) ya que tiene el bloque válido. El bloque pasa a estado modificado en la cache de N1.	**N1) Modificado** **N2) Inválido** **N3) Inválido**

N1) Modificado **N2) Inválido** **N3) Inválido**	P2 escribe en k	- N2 genera petición de lectura con acceso exclusivo de k (PtLecEx(k)) - N1 observa PtLecEx(k) y, como tiene el bloque en estado modificado, genera respuesta con el bloque RpBloque (k) inhibiendo la respuesta de memoria, y además, como el paquete pide exclusividad, invalida su copia del bloque. - N2 recoge RpBloque(k). El bloque entra en estado modificado en la cache de N2	**N1) Inválido** **N2) Modificado** **N3) Inválido**
N1) Inválido **N2) Modificado** **N3) Inválido**	P3 escribe en k	- N3 genera petición de lectura con acceso exclusivo de k PtLecEx(k) - N2 observa PtLecEx(k) y, como tiene el bloque en estado modificado, genera respuesta con el bloque RpBloque (k), y además, como el paquete pide exclusividad, invalida su copia del bloque. - N3 recoge RpBloque(k). El bloque entra en estado modificado en la cache de N3	**N1) Inválido** **N2) Inválido** **N3) Modificado**


CC: Controlador Cache
CM: Controlador de Memoria
Directorio cache
1 0
Estado bloque cache
Nodo de cómputo
P
CC
C
E/S
Conmu-tador
CM
Memoria
conexión a otros nodos de cómputo
Memoria
bloque
bloque
bloque
Directorio
C0 C1 ... CN-1
1 0 ... 1 0
Estado bloque memoria
Vector de bits con información sobre cachés con copia

Ejercicio 2. Para un multiprocesador de memoria distribuida con 8 nodos se quiere implementar un protocolo para mantenimiento de coherencia basado en directorios. Suponiendo que se necesitan un bit de estado para un bloque en el directorio de memoria principal y que el tamaño de una línea de cache es de 64 bytes, calcular el porcentaje del tamaño de memoria principal que supone el tamaño del directorio de vector de bits completo.

Solución

Datos del ejercicio:

- Tamaño Línea de Cache (bloque de memoria): 64 bytes = TLC = 2^6
- 1 bit de estado por bloque en el directorio

Directorio de vector de bits completo

Tamaño_por_nodo = Nº_de_bloques_memoria_nodo × Espacio_por_bloque

TMN: Tamaño Memoria principal por Nodo

Teniendo en cuenta que hay 8 nodos y un bit de estado, se necesitan 9 bits por entrada del directorio.

$$Tamaño\_directorio\_por\_nodo = \frac{TMN}{TLC} \times (8b + 1b)$$

$$\%\_de\_TMN = \frac{\frac{TMN}{TLC} \times 9b}{TMN} \times 100 = \frac{9b}{TLC} \times 100 = \frac{9b}{64 \times 8b} \times 100 \approx 1{,}76\%$$

Ejercicio 3. Suponga que en un CC-NUMA de red estática de 4 nodos (N0-N3) se implementa un protocolo MSI basado en directorios sin difusión con dos estados en el directorio (válido e inválido). Cada nodo tiene 8 GBytes de memoria y una línea de cache supone 64 Bytes. Considere que el directorio utiliza vector de bits completo. **(a)** Calcule el tamaño del directorio en bytes. **(b)** Indique cual sería el contenido del directorio, las transiciones de estados (en cache y en el directorio) y la secuencia de paquetes generados por el protocolo de coherencia en los siguientes accesos sobre una dirección D que se encuentra en la memoria del nodo 3 (inicialmente D no está en ninguna cache):

1. Lectura generada por el procesador del nodo 1
2. Escritura generada por el procesador del nodo 1
3. Lectura generada por el procesador del nodo 2
4. Lectura generada por el procesador del nodo 3
5. Escritura generada por el procesador del nodo 0

Solución

Datos del ejercicio

- Tamaño Memoria principal por Nodo: 8GB = TMN
- Tamaño Línea de Cache (bloque de memoria): 64 B = TLC
- 1 bits de estado por bloque en el directorio ya que hay que codificar dos estados (válido, inválido).
- Se accede a una dirección de la memoria del nodo 5 cuyo bloque no se encuentra en ninguna cache, luego debe estar actualizado en memoria principal.

(a)

$$Tamaño\_por\_nodo = \frac{TMN}{TLC} \times (1b + 4b) = \frac{2^{33} B}{2^{6} B} \times (5)b = (2^{27} \times 5b)$$

2^{27}x5 bits / (2^3bits/Byte)= 2^{24}x5 Bytes=5x2^4 MBytes = 80 Mbytes

Intervienen los nodos N0, N1, N2 y N3. V es Válido e I inválido. BD denota el bloque de la dirección D.

ESTADO INICIAL	EVENTO	ACCIÓN	ESTADO SIGUIENTE
N0) Inválido **N1) Inválido** **N2) Inválido** **N3) Inválido** **D) Local** V | | | | |	P1 lee D	1. N1 envía petición de lectura del bloque BD (PtLec(BD)) a N3 2. N3 recibe PtLec(BD). Como tiene el bloque BD en estado Válido, (1) envía paquete de respuesta con el bloque a N1 (RpBloque(BD)) y (2) pone el bit de N1 en el directorio a 1. 3. N1 recibe RpBloque(BD) y pone el bloque en cache en estado Compartido	**N0) Inválido** **N1) Compartido** **N2) Inválido** **N3) Inválido** **D) Válido** V | | 1 | | |
N0) Inválido **N1) Compartido** **N2) Inválido** **N3) Inválido** **D) Válido** V | | 1 | | |	P1 escribe en D	1. N1 envía petición de acceso exclusivo para BD (PtEx(BD)) a N3. 2. N3, cuando recibe PtEx(BD), (1) pasa BD a estado Inválido y (2) envía paquete de respuesta a N1 confirmando invalidación (RpInv(BD)). 3. N1, recibida la respuesta, modifica el bloque y lo pasa a estado Modificado.	**N0) Inválido** **N1) Modificado** **N2) Inválido** **N3) Inválido** **D) Inválido** I | | 1 | | |

N0) Inválido **N1) Modificado** **N2) Inválido** **N3) Inválido** **D) Inválido** I | | 1 | |	P2 lee D	1. N2 envía PtLec(BD) a N3 porque no tiene BD. 2. N3, como tiene BD en estado Inválido: (1) reenvía (RvPetLec(BD)) la petición al nodo N1 (que según el directorio tiene copia válida del bloque), y (2) pone en la entrada del bloque en el directorio estado pendiente de Válido y activa el bit de N2. 3. N1 recibe RvPetLec(BD) y: (1) envía a N3 un paquete de respuesta con el bloque (RpBloque(BD)), y (2) pasa BD en su cache a Compartido. 4. N3 recibe la respuesta de N1 (RpBloque(BD)) y: (1) responde con el bloque a N2 (RpBloque(BD)) y (2) activa el bit de N2 y pone el estado del bloque en el directorio a Válido 5. N2, cuando recibe RpBloque(BD), introduce el bloque en su cache en estado Compartido	**N0) Inválido** **N1) Compartido** **N2) Compartido** **N3) Inválido** **D) Válido** V | | 1 | 1 |
N0) Inválido **N1) Compartido** **N2) Compartido** **N3) Inválido** **D) Válido** V | | 1 | 1 |	P3 lee D	N3 lee BD de su propia memoria, lo mete en su cache en estado Compartido y activa el bit de copia de N3 en el directorio	**N0) Inválido** **N1) Compartido** **N2) Compartido** **N3) Compartido** **D) Válido** V | | 1 | 1 | 1

N0) Inválido **N1) Compartido** **N2) Compartido** **N3) Compartido** **D) Válido** V | | 1 | 1 | 1	P0 escribe en D	1. N0 envía a N3 petición de lectura de BD con acceso exclusivo PtLecEx(BD) 2. N3 recibe PtLecEx(BD) y, como tiene el bloque en estado Válido: (1) envía los paquetes RvInv(BD) a los nodos que, según el directorio, tienen copia del bloque, (2) pone en el directorio el estado de BD en pendiente de Inválido. 3. N1, N2 y N3, cuando reciben RvInv(BD): (1) invalidan su copia de BD y (2) responden a N3 confirmando la invalidación RpInv(BD). 4. N3, cuando recibe todas las RpInv(BD): (1) envía respuesta con el bloque a N0 (espera a todas las invalidaciones para garantizar un orden en los accesos a BD y así garantizar coherencia) y (2) activa el bit de copia de N0 en el directorio y pone el estado de BD en el directorio a Inválido 5. N0 escribe BD en cuanto recibe RpBloque de N3 y lo pone en estado Modificado	**N0) Modificado** **N1) Inválido** **N2) Inválido** **N3) Inválido** **D) Inválido** I | 1 | | | |

Aviso

“Con motivo de la suspensión temporal de la actividad docente presencial en la UGR, se informa de las condiciones de uso de la aplicación de videoconferencia que a continuación se va a utilizar:

1. La sesión va a ser grabada con el objeto de facilitar al estudiantado, con posterioridad, el contenido de la sesión docente.

2. Se recomienda a los asistentes que desactiven e inhabiliten la cámara de su dispositivo si no desean ser visualizados por el resto de participantes.

3. Queda prohibida la captación y/o grabación de la sesión, así como su reproducción o difusión, en todo o en parte, sea cual sea el medio o dispositivo utilizado. Cualquier actuación indebida comportará una vulneración de la normativa vigente, pudiendo derivarse las pertinentes responsabilidades legales.”

Ejercicio 4. Supongamos que se va a ejecutar en paralelo el siguiente código (inicialmente *x* e *y* son 0):

P1	**P2**
x=1;	y=1;
x=2;	y=2;
print y ;	print x ;

Qué resultados se pueden imprimir si (considere que el compilador no altera el código):

(a) Se ejecutan P1 y P2 en un multiprocesador con consistencia secuencial.

(b) Se ejecutan en un multiprocesador basado en un bus que garantiza todos los órdenes excepto el orden W→R. Esto es debido a que los procesadores tienen buffer de escritura, permitiendo el procesador que las lecturas en el código que ejecuta adelanten a las escrituras que tiene su buffer. Obsérvese que hay varios posibles resultados.

Solución

El compilador no altera ningún orden garantizado ya que se supone, según el enunciado, que no altera el código.

(a) Si P1 es el primero que imprime puede imprimir 0, 1 o 2, pero P2 podrá imprimir sólo 2. Esto es así porque se mantiene orden secuencial y, por tanto, si P1 ha leído "y", ha asignado ya a "x" un 2. Si P2 es el primero que imprime podrá imprimir 0, 1 o 2, pero entonces P1 sólo puede imprimir 2. Combinaciones:

P1	P2
0	2
1	2
2	2
2	0
2	1

(b) Si no se mantiene el orden W→R además de los resultados anteriores, los dos procesos pueden imprimir:

P1	P2
1	1
0	1
0	2
1	0
2	0
0	0

ya que no se asegura que cuando un proceso ejecuta la lectura de la variable que imprime print haya ejecutado las instrucciones anteriores que escriben en x (P1) o en y (P2). P1 puede imprimir 1 o 2 o 0, y P2 1 o 2 o 0. Todas las combinaciones son posibles.

Ejercicio 5. Supongamos que se va a ejecutar en paralelo el siguiente código (inicialmente *x* e *y* son 0):

P1	**P2**
`x=30;`	`while (flag==0) {};`
`y=40;`	`r1=x;`
`flag=1;`	`r2=y;`

Qué datos puede obtener P2 en r1 y r2 si (considere que el compilador no altera el código):

(a) Se ejecutan P1 y P2 en un multiprocesador con consistencia secuencial.
(b) Se ejecutan en un multiprocesador con consistencia de liberación. Razone su respuesta.

El compilador no altera ningún orden garantizado ya que se supone, según el enunciado, que no altera el código.

(a) Si se implementa consistencia secuencial en r1 se almacena 30 y en r2 40.

(b) Si se implementa consistencia de liberación se pueden dar los siguientes resultados

r1	r2
0	0
0	40
30	0
30	40

RAZONAMIENTO:

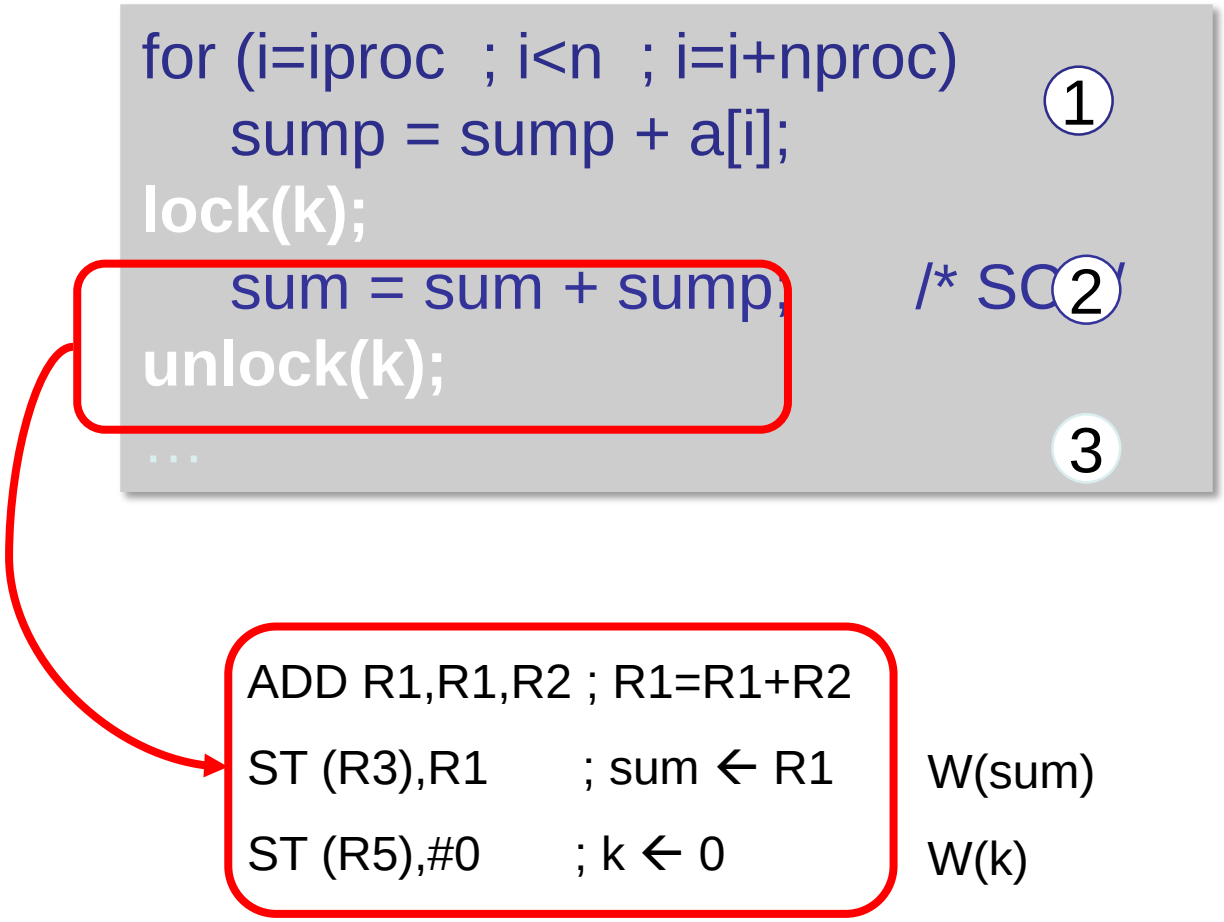
1) Al no garantizarse el orden entre accesos de escritura (W->W), P1 podría escribir en flag un 1 antes de escribir 30 en x o 40 en y (obsérvese que la escritura y=40 podría también adelantar a x=30). Por lo que P2 podría leer en flag un 1 y, por tanto, salir del bucle, antes de que en x o en y pudiera ver los valores que escribe P1 en estas variables (vería entonces 0).

2) Al no garantizarse el orden entre accesos de lectura (R->R), si se permite ejecutar lecturas cuya ejecución depende de una condición de salto antes de verificar que se cumple la condición (ejecución especulativa), en P2 se podría, en la última iteración del bucle, adelantar la lectura de x o la lectura de y o ambas a la de flag. En este caso P2 podría entonces leer de forma especulativa los valores de x e y que tienen en su cache antes de que P1 los modifique y modifique flag. En estas circunstancias podría obtenerse en r1 o en r2 o en ambos un 0.

Ejercicio 6. Se quiere implementar un cerrojo simple en un multiprocesador SMP basado en procesadores de la línea x86 de Intel, en particular, procesadores Intel Core.

(a) Teniendo en cuenta el modelo de consistencia de memoria que ofrece el hardware de este multiprocesador ¿podríamos implementar la función de liberación del cerrojo simple usando “mov k, 0”, siendo k la variable cerrojo? Razone su respuesta.

(b) ¿Cómo se debería implementar la función de liberación de un cerrojo simple si se usan procesadores Itanium? Razone su respuesta.



X86: solo relaja W → R, pero no relaja W → W

Por lo tanto, no hay problema

```
for (i=iproc  ; i<n  ; i=i+nproc)
     sump = sump + a[i];
lock(k);
     sum = sum + sump;        /* SC
unlock(k);
...
```

1

2

3

```
ADD R1,R1,R2 ; R1=R1+R2
ST (R3),R1       ; sum ← R1
ST (R5),#0       ; k ← 0
```

La primitiva de sincronización unlock() debe implementarse de otra forma: asegurando que los accesos a memoria previos se han completado (instrucción ST.REL)

No se garantiza el funcionamiento correcto de la sección crítica

```
ST (R5),#0       ; k ← 0
ADD R1,R1,R2 ; R1=R1+R2
ST (R3),R1       ; sum ← R1
```

Los Itanium relajan todos los órdenes

Ejercicio 7. Se ha ejecutado el siguiente código en un multiprocesador con un modelo de consistencia que no garantiza ni W->R ni W->W (garantiza el resto de órdenes):

```
sump = 0;
for (i=ithread  ; i<8  ; i=i+nthread) {
        sump = sump + a[i];
    }
    while (Fetch_&_Or(k,1)==1)  {};
    sum = sum + sump;
    k=0;
```

(a) Indique qué se puede obtener en sum si se suma la lista a={1,2,3,4,5,6,7,8}. k y sum son variables compartidas que están inicialmente a 0 (el resto de variables son privadas), nthread = 3 y ithread es el identificador del thread en el grupo (0,1,2). Si hay varios posibles resultados, se tienen que dar todos ellos. Justifique su respuesta.

(b) ¿Qué resultados se pueden obtener si lo único que no garantiza el modelo de consistencia es el orden W->R? Justifique su respuesta.

0 (1) 1 (2) 2 (3)

3 (4) 4 (5) 5 (6)

6 (7) 7 (8)

12 15 9

Thread 0 1 2

```
sump = 0;
for (i=ithread  ; i<8  ; i=i+nthread) {
    sump = sump + a[i];
}
while (Fetch_&_Or(k,1)==1)  {};
sum = sum + sump;
k=0;
```

No hay problemas con Fetch_&_Or(k,1) porque es una operación atómica que contiene R (también tiene W) y ni las R ni las W pueden adelantar a R anteriores en el orden del programa.

Se podría liberar el cerrojo antes de que se hubiera completado la escritura de la suma parcial

Se podrían observar distintas combinaciones de sumas parciales

	resultado	comentario
(R0W0)(R1W1)(R2W2)	**12+15+9=36**	Si no hay problemas debido a que la escritura de liberación adelante a los accesos a `sum` anteriores
RxRxRxWxWxW0	**12**	Si los tres flujos de control llegan a leer el valor 0 inicial de `sum` y el último que escribe en `sum` tras actualizar su valor es el thread 0
RxRxRxWxWxW1	**15**	Si los tres flujos de control llegan a leer el valor 0 inicial de `sum` y el último que escribe en `sum` es el thread 1
RxRxRxWxWxW2	**9**	Si los tres flujos de control llegan a leer el valor 0 inicial de `sum` y el último que escribe en `sum` es el thread 2
R0W0RxRxW2W1 R0W0RxRxW1W2	**12+15=27 o 12+9=21**	Si el flujo de control 0 ha logrado acumular primero sin problemas y el 1 y el 2 leen el valor que tiene `sum` tras la acumulación de 0. Si esto ocurre entonces 1 y 2 acceden al mismo valor, 12, le acumulan cada uno lo que han calculado y escriben el resultado de la acumulación en `sum`. El resultado será 27 si el último que escribe es 1 y 21 si el último que escribe es 2.
R1W1RxRxW2W0 R1W1RxRxW0W2	**15+12=27 o 15+9=24**	Si el flujo de control 1 ha logrado acumular primero sin problemas y el 0 y el 2 acceden al valor que tiene `sum` justo tras la acumulación de 1. Si esto ocurre entonces 0 y 2 acceden al mismo valor, 15, le acumulan cada uno lo que han calculado y escriben el resultado de la acumulación en `sum`. El resultado será 27 si el último que escribe es 0 y 24 si el último que escribe es 2.
R2W2RxRxW1W0 R2W2RxRxW0W1	**9+12=21 o 9+15=24**	Si el flujo de control 2 ha logrado acumular primero sin problemas y el 0 y el 1 acceden al valor que tiene `sum` justo tras la acumulación de 2. Si esto ocurre entonces 0 y 2 acceden al mismo valor, 9, le acumulan cada uno lo que han calculado y escriben el resultado de la acumulación en `sum`. El resultado será 21 si el último que escribe es 0 y 24 si el último que escribe es 1.

(b) Si se garantiza el orden W → W no hay problema: **k = 0** se hace después de **sum=sum+sump**

Ejercicio 11

```
Barrera(id, num_procesos)
{
  band_local= !(band_local)
  while (fetch_&_or(k,1)==1) {};                    (R,W) atómica
            cont_local = ++bar[id].cont;            R seguida de W
  k=0;                                              W
  if (cont_local == num_procesos)  {
      bar[id].cont=0;
      bar[id].band=band_local;
  }
  else while (bar[id].band != band_local) {};
}
```

(a) No se garantiza W->R: Las escrituras (W) no pueden adelantar a escrituras previas. No habría problema

(b) Da problemas en la escritura en k=0 (se ha visto antes). Tener en cuenta que una escritura puede adelantar a lecturas o escrituras previas pero las lecturas no adelantan a escrituras previas que van al mismo dato (dependencia RAW)

CASO a)

----- lock (acceso atómico: se adelantan las dos o ninguna)

R(k) fetch_&_or (k,1)

W(k)

R(bar[id].cont)

W(bar[id].cont)

W(k) ---- unlock

W(bar[id].cont)

W(bar[id].band)

NO EXISTE W->R:

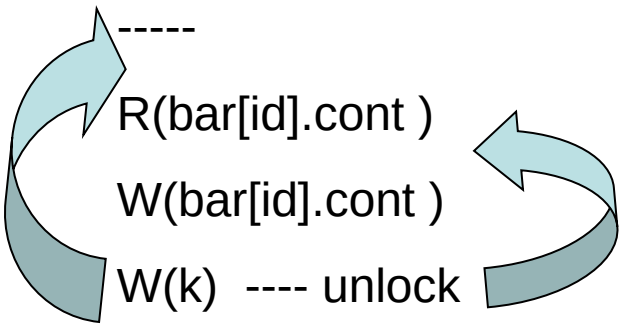
Se mantiene el orden secuencial de los accesos

CASO b)

----- lock (acceso atómico: se adelantan las dos o ninguna)

R(k) fetch_&_or (k,1)

W(k)



R(bar[id].cont)

W(bar[id].cont)

W(k) ---- unlock

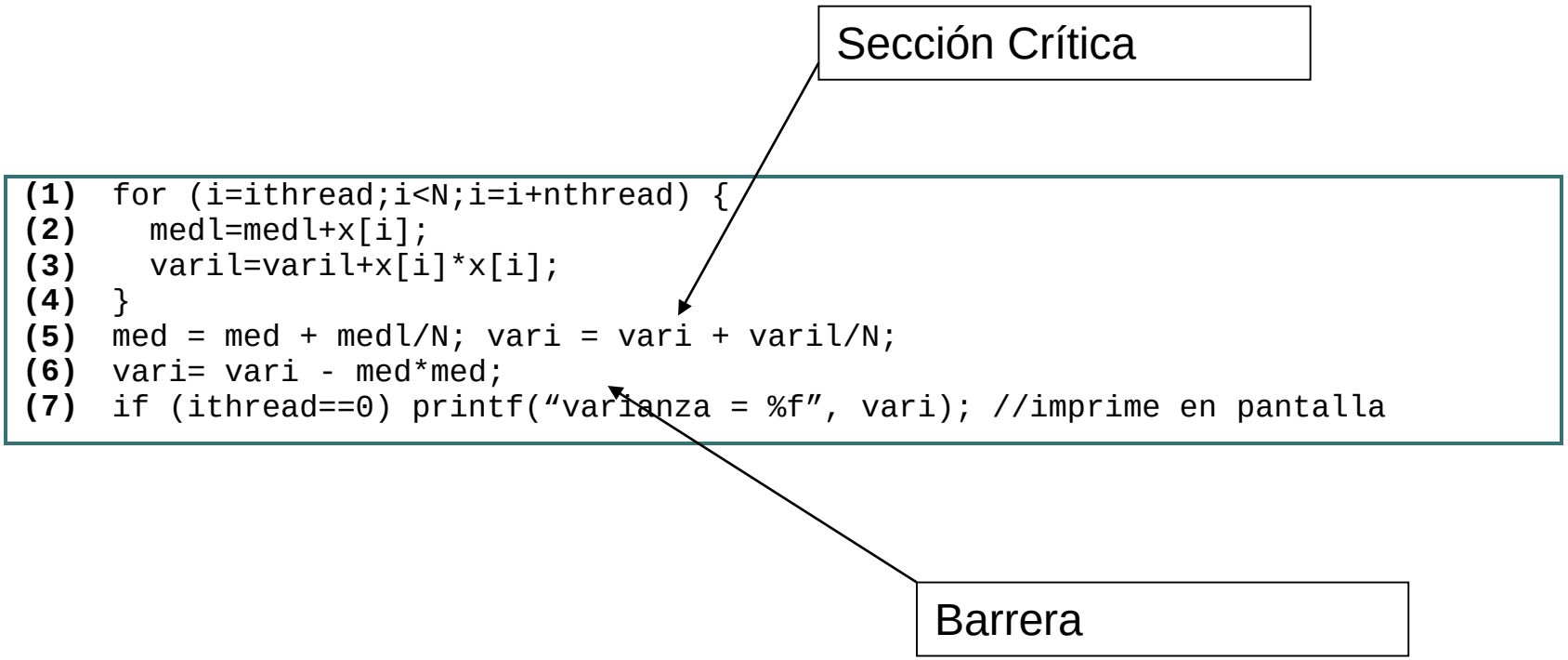
W(bar[id].cont)

W(bar[id].band)

EXISTEN W->W y R->W:

La apertura de la barrera puede hacerse antes de que concluya la sección crítica

Ejercicio 12



```
(1)  for (i=ithread;i<N;i=i+nthread) {
(2)    medl=medl+x[i];
(3)    varil=varil+x[i]*x[i];
(4)  }
     while (test_&_set(k1)) {};   //lock(k1)
(5)  med = med + medl/N; vari = vari + varil/N;
     k1=0;                        //unlock(k1)

     bandera_local= !(bandera_local)    //se complementa la bandera local
     while (test_&_set(k2)) {};     //lock(k2)
     cont_local = bar[id].cont+1;    //cont_local es local
     k2=0;                          //unlock(k2)
     if (cont_local ==num_procesos-1)  {
           bar[id].cont=0;                          //se   hace   0   el   contador
     asociado a la barrera
           bar[id].bandera= bandera_local;   // libera procesos en espera
     }
     else  while (bar[id].bandera!= bandera_local) {};
(6)
(7)  vari= vari - med*med;
     if (ithread==0) printf("varianza = %f", vari); //imprime en pantalla
```

```
(1)  for (i=ithread;i<N;i=i+nthread) {
(2)    medl=medl+x[i];
(3)    varil=varil+x[i]*x[i];
(4)  }
(5)  fetch_&_add(med,medl/N); fetch_&_add(vari,varl/N);

     bandera_local= !(bandera_local)    //se complementa la bandera local
     while (test_&_set(k2)) {};     //lock(k2)
     cont_local = fetch_&_add(bar[id].cont,1);   //cont_local es local
     if (cont_local==num_procesos-1)  {
            bar[id].cont=0;                          //se   hace   0   el   contador
     asociado a la barrera
            bar[id].bandera= bandera_local;   // libera procesos en espera
     }
     else  while (bar[id].bandera!= bandera_local) {};
(6)
(7)
     vari= vari - med*med;
     if (ithread==0) printf("varianza = %f", vari); //imprime en pantalla
```

```
(1)  for (i=ithread;i<N;i=i+nthread) {
(2)    medl=medl+x[i];
(3)    varil=varil+x[i]*x[i];
(4)  }
(5)  do
        a = med;
        b = a + medl/N;
        compare&swap(a,b,med);
     while (a!=b);
     do
        a = vari;
        b = a + varil/N;
        compare&swap(a,b,vari);
     while (a!=b);

     bandera_local= !(bandera_local)    //se complementa la bandera local
     while (test_&_set(k2)) {};     //lock(k2)
     do
        cont_local = bar[id].cont;
        b = cont_local + 1;
        compare&swap(cont_local,b,bar[id].cont);
     while (cont_local!=b);
     cont_local = fetch_&_add(bar[id].cont,1);   //cont_local es local
     if (cont_local==num_procesos-1)  {
           bar[id].cont=0;                          //se   hace   0   el   contador
     asociado a la barrera
           bar[id].bandera= bandera_local;   // libera procesos en espera
     }
     else  while (bar[id].bandera!= bandera_local) {};
(6)
(7)  vari= vari - med*med;
     if (ithread==0) printf("varianza = %f", vari); //imprime en pantalla
```