Soluciones a ejercicios de multiprocesadores

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Javier García Blas

Computer Architecture
ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

1. Ejercicios de examen

Exercise 1 *Examen de junio de 2015.*

Sea un computador con dos procesadores, cada uno con su memoria caché privada que usan política de post-escritura a memoria principal. Inicialmente todas las entradas de las cachés se encuentran marcadas como inválidas. La posición de memoria correspondiente a la variable **A**, contiene inicialmente el valor **255**.

En este computador se ejecuta la siguiente secuencia de operaciones:

Tiempo	Procesador 1	Procesador 2
1	**lw $t0, A**	
2		**lw $t1, A**
3	**sw $zero, A**	
4		**lw $t2, A**

Indique para cada una de estas operaciones el estado y el valor de la dirección de memoria **A** tanto en las cachés como en memoria principal.

Tiempo	Estado caché 1	Valor caché 1	Estado caché 2	Valor caché 2	Valor memroia
0	I	–	I	–	255
1					
2					
3					
4					

Solution 1

La Tabla 1 muestra los estados y valores:

Exercise 2 *Examen de enero de 2015.*

Dados los siguientes fragmentos de código que ejecutan 3 hilos empleando el protocolo de coherencia MSI.

Tiempo	Estado caché 1	Valor caché 1	Estado caché 2	Valor caché 2	Valor memroia
0	I	–	I	–	255
1	Compartido	255	Inválido	–	255
2	Compartido	255	Compartido	255	255
3	Exclusivo	0	Inválido	255	255
4	Compartido	0	Compartido	0	0

Cuadro 1: Solución del ejercicio 1.

```
// Código del hilo 0
for(i=0;i<16;i++){
  a[i]= 16;
}

// Código del hilo 1
tmp=0;
for(i=0;i<16;i++){
  a[i]=tmp;
  tmp+=a[i];
}

// Código del hilo 2
cnt=0;
for(i=0;i<4;i++){
  cnt+=b[i];
}
```

El computador tiene una arquitectura CC-NUMA con:

- 2 procesadores de 32 bits con único nivel de memoria caché que es privado a cada procesador y tiene una correspondencia directa. El tamaño de línea caché es de 16 bytes.
- Las memorias caché están inicialmente vacías.
- Los hilos 0 y 2 se ejecutan en el procesador 0 mientras que el hilo 1 se ejecuta en el procesador 1.
- Las variables **i**, **tmp** y **cnt** se almacenan en registros (no se almacenan en memoria).
- El bloque que contiene **a[0]** está asociado a la misma línea caché que el bloque que contiene **b[0]**.

Se pide rellenar las siguientes tablas y justificar la respuesta para los siguientes apartados:

1. Partiendo de la situación inicial, indicar en la siguiente tabla las transiciones de estado del bloque que almacena **a[0]** cuando se ejecuta primero el hilo 0 completamente y a continuación el hilo 1. Indique el tráfico de bus asociado a cada hilo.
   - **Nota**: en el caso de haber varias transiciones asociadas a un hilo, indique todas ellas.

Código	Transición P0	Transición P1	Señales de bus
Hilo 0			
Hilo 1			

2. Partiendo de la situación inicial, indicar en la siguiente tabla las transiciones de estado del bloque que almacena **a[0]** cuando se ejecuta primero el hilo 1 completamente y a continuación el hilo 0 Indique el tráfico de bus asociado a cada hilo.
   - **Nota**: en el caso de haber varias transiciones asociadas a un hilo, indique todas ellas.

Código	Transición P0	Transición P1	Señales de bus
Hilo 0			
Hilo 1			

3. Partiendo de la situación inicial, indicar en la siguiente tabla las transiciones de estado del bloque que almacena **a[0]** cuando se ejecuta primero el hilo 0 completamente, a continuación el hilo 2 completamente y finalmente el hilo 1 Indique el tráfico de bus asociado a cada hilo.

   - **Nota**: en el caso de haber varias transiciones asociadas a un hilo, indique todas ellas.

Código	Transición P0	Transición P1	Señales de bus
Hilo 0			
Hilo 1			
Hilo 2			

4. Para cada uno de los escenarios anteriores, indique el número de fallos caché existente para cada proceso.

Escenario	Fallos caché P0	Fallos caché P1
Hilo 0 → hilo 1		
Hilo 1 → hilo 0		
Hilo 0 → hilo 2 → hilo 1		

Solution 2

Apartado 1 Se muestran las transiciones en la Tabla 2.

Código	Transición P0	Transición P1	Señales de bus
Hilo 0	I → E	I	Write miss
Hilo 1	E → I	I → E , E → E	Write miss; Write-back block

Cuadro 2: Transiciones de apartado 1 de ejercicio 2

Apartado 2 Se muestran las transiciones en la Tabla 3.

Código	Transición P0	Transición P1	Señales de bus
Hilo 1	I	I → E ; E → E	Write miss
Hilo 0	I → E	E → I	Write miss; Write-back block

Cuadro 3: Transiciones de apartado 2 de ejercicio 2

Apartado 3 Se muestran las transiciones en la Tabla 4.

Cada bloque tiene 4 palabras. Por lo que los hilos 0 y 1 acceden a 4 bloques y mientras que el hilo 2 accede a 1 bloque. En el hilo 1 la primera línea causa fallo caché mientras que la segunda, acierto.

Código	Transición P0	Transición P1	Señales de bus
Hilo 0	I → E (almacena **a[0]**)	I	Write miss
Hilo 2	E → S (almacena **b[0]**)	I	Read miss; write back block
Hilo 1	S → S (**b[0]**)	I → E ; E → E	Write miss

Cuadro 4: Transiciones de apartado 3 de ejercicio 2

Escenario	Fallos caché P0	Fallos caché P1
hilo 0 → hilo 1	4	4
hilo 1 → hilo 0	4	4
hilo 0 → hilo 2 ->hilo 1	4 + 1	4

Cuadro 5: Transiciones de apartado 4 de ejercicio 2

Apartado 4 Se muestran las transiciones en la Tabla 5.

Exercise 3 *Examen de enero de 2014.*

Sea un multiprocesador con arquitectura de memoria compartida simétrica basado en bus con protocolo de espionaje o snooping. Cada procesador tiene una caché privada cuya coherencia se mantiene usando el protocolo MSI. Cada caché utiliza correspondencia directa y tiene cuatro bloques cada uno con dos palabras. Esta caché utiliza como campo de etiqueta la dirección de memoria completa.

Las siguientes tablas muestran el estado de cada memoria, con la palabra menos significativa a la izquierda.

Procesador P0			
Bloque	**Estado**	**Etiqueta**	**Datos**
B0	I	0x00100700	0x00000000 0x7FAABB11
B1	S	0x00100708	0x00000000 0x00001234
B2	M	0x00100710	0x00000000 0x0077AABB
B3	I	0x00100718	0x00000000 0x7FAABB11

Procesador P1			
Bloque	**Estado**	**Etiqueta**	**Datos**
B0	I	0x00100700	0x00000000 0x7FAABB11
B1	M	0x00100728	0x00000000 0xFF000000
B2	I	0x00100710	0x00000000 0xEEEE7777
B3	S	0x00100718	0x00000000 0x7FAABB11

Procesador P2			
Bloque	**Estado**	**Etiqueta**	**Datos**
B0	S	0x00100720	0x00000000 0x1111AAAA
B1	S	0x00100708	0x00000000 0X00001234
B2	I	0x00100710	0x00000000 0x7FAABB11
B3	I	0x00100718	0x00001234 0x1111AABB

Para cada uno de los apartados que a continuación se presentan, parta de la situación inicial del problema, sin tener en cuenta los cambios de los apartados anteriores. Indique los cambios que se producen en las cachés. En el caso de las lecturas, indique además cuál es el valor efectivamente leído.

1. P2: write 0x00100708, 0xFFFFFFFF

2. P2: read 0x00100708
3. P2: read 0x00100718

En cada apartado deberá rellenar una tabla con el siguiente formato, justificando la respuesta:

Procesador	**Bloque**	**Estado anterior**	**Estado actual**	**Etiqueta**	**Datos**	

Solution 3

Apartado 1 Se produce una escritura en el bloque B1 de la caché de P2 que se encuentra en estado S. Esto produce los siguientes cambios en P2:

- Se pasa a estado exclusivo (M).
- Se coloca una invalidación en el bus.

La invalidación es ignorada por el procesador P1, pero es tratada por el procesador P0.
En P0 se producen los siguientes cambios:

- Se pasa el estado a inválido (I).

En este caso no se producen modificaciones en la memoria principal.
Los estados en los distintos procesadores se muestran en la Tabla 6.

Procesador P0				
Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	I	0x00100708	0x00000000	0x00001234
B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11
Procesador P1				
Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11
Procesador P2				
Bloque	Estado	Etiqueta	Datos	
B0	S	0x00100720	0x00000000	0x1111AAAA
B1	M	0x00100708	0xFFFFFFFF	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	I	0x00100718	0x00001234	0x1111AABB

Cuadro 6: Estados de caché correspondientes al apartado 1 del ejercicio 3.

Las transiciones de estados y las correspondientes modificaciones se muestran en la Tabla **??**.

Procesador	Bloque	Estado Anterior	Estado Nuevo	Etiqueta	Datos	
P2	B1	I	M	0X00100708	0XFFFFFFFF	0X00001234
P0	B1	S	I	0X00100708	0X00000000	0X00001234

Cuadro 7: Transicones de estados y modificaciones correspondientes al apartado 1 del ejercicio 3.

Apartado 2 Se produce una lectura en el bloque B1 de la caché P2 que se encuentra en el estado S. Se trata de un acierto y no se produce ningún cambio en las cachés o en memoria.

El valor leído es: 0x00000000

Apartado 3 Se produce un fallo de lectura la caché de P2 para el bloque B3 que está en estado inválido (I). Por tanto se coloca el fallo de lectura en el bus y se pasa a estado compartido (S).

El procesador P0, tiene este bloque en estado inválido e ignora el fallo de lectura.

El procesador P1, tiene este bloque en estado compartido (S) y continúa en este estado.

El valor leído es 0x00000000

Los estados en los distintos procesadores se muestran en la Tabla 8.

Procesador P0				
Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	S	0x00100708	0x00000000	0x00001234
B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11
Procesador P1				
Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11
Procesador P2				
Bloque	Estado	Etiqueta	Datos	
B0	S	0x00100720	0x00000000	0x1111AAAA
B1	S	0x00100708	0x00000000	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	S	0x00100718	0x00000000	0x7FAABB11

Cuadro 8: Estados de caché correspondientes al apartado 3 del ejercicio 3.

Las transiciones de estados y las correspondientes modificaciones se muestran en la Tabla **??**.

Procesador	Bloque	Estado Anterior	Estado Nuevo	Etiqueta	Datos	
P2	B3	I	S	0X00100718	0X00000000	0X7FAABB11

Cuadro 9: Transicones de estados y modificaciones correspondientes al apartado 3 del ejercicio 3.

Exercise 4 *Examen de junio de 2014.*

Sea un procesador con arquitectura **Intel P6 o posterior**, en el que se tienen dos variables (**z** y **t**) que inicialmente tienen el valor **42**. En dicho procesador dos hilos ejecutan concurrentemente.

El hilo 1 ejecuta:

```
mov [_z], 1
mov r1, [_z]
mov r2, [_t]
```

El hilo 2 ejecuta:

```
mov [_t], 1
mov r3, [_t]
mov r4, [_z]
```

¿Es posible que al final de la ejecución del hilo 2 los registros **r2** y **r4** tengan ambos el valor de **42**? Justifique su respuesta.

Solution 4

Si es posible porque las escrituras **pueden percibirse en distinto orden por cada procesador**. De esta manera en el hilo 1, puede tenerse **r1=1** y **r2=42**, mientras que en el hilo 2, puede tenerse **r3=1** y **r4=42**.

Exercise 5 *Examen de junio de 2014.*

Sea un multiprocesador con arquitectura de memoria compartida simétrica basado en bus con protocolo de espionaje o snooping. Cada procesador tiene una caché privada cuya coherencia se mantiene usando el protocolo MSI. Cada bloque caché tiene una única palabra.

La siguiente tabla muestra el estado inicial de cuatro variables distintas en cada una de las cachés.

	Estado inicial de las variables			
Procesador	**A**	**B**	**C**	**D**
P0	Shared	Exclusive	Shared	Shared
P1	Invalid	Invalid	Invalid	Shared
P2	Invalid	Invalid	Shared	Shared

La siguiente tabla muestra el estado final de estas variables tras realizar una serie de accesos a memoria.

	Estado final de las variables			
Procesador	**A**	**B**	**C**	**D**
P0	Invalid	Invalid	Invalid	Shared
P1	Invalid	Invalid	Shared	Exclusive
P2	Exclusive	Exclusive	Invalid	Shared

Se pide:

- Para cada variable (A, B, C y D) describir de forma justificada el/los accesos realizados y el/los procesos involucrados que han permitido alcanzar el estado final.
  - **Nota1**: para alcanzar el estado final puede ser suficiente un solo acceso o una secuencia de accesos.
  - **Nota2**: puede existir un estado final que sea inalcanzable (es decir, que no tiene solución).
- Para cada caso anterior describir el tráfico de bus generado

Solution 5

- Variable **A**:
  - **P2** escribe **A** (estado a *exclusive*) invalidando la copia en la caché de **P0**.
  - **P2** genera un fallo de escritura en el bus que lo captura **P0** el cual no genera tráfico.
- Variable **B**:
  - **P2** escribe **B** (estado a exclusive) invalidando la copia en la caché de **P0**
  - **P2** genera un write miss en el bus que lo captura **P0** el cual realiza un write-back del bloque a la caché de **P2**
- Variable **C**:
  - **P1** escribe **B** (estado a exclusive) invalidando las otras dos copias.
  - **P1** genera un write miss en el bus que lo captura **P0** y **P1** los cuales no generan tráfico.
  - **P1** lee o escribe **D** otra variable cuyo bloque ocupa la misma línea caché que **C** generando un fallo de conflicto. Esto hace que el bloque de **C** se reemplace y como ha sido modificado se escribe en memoria. El bloque de **C** no está presente en caché (equivalente a estado *invalid*).
  - Esta acción de **P1** genera de tráfico realizando el *write-back* del bloque en memoria.
  - **P1** lee **C**. Genera un fallo de escritura y pasa a estado compartido.
- Variable **D**:
  - El estado final no se puede alcanzar porque un bloque no puede estar exclusivo y compartido a la vez.

Exercise 6 *Examen de enero de 2013.*

Un computador dispone de un procesador Intel Core Dúo de 32 bits con dos núcleos (cores) con la configuración mostrada en la figura. El computador utiliza el protocolo de coherencia de caché MSI, el cual se aplica en todos los niveles de memoria caché. El tamaño de bloque es de 16 Bytes y el de cada palabra de 4 Bytes, el cual aplica a todos los niveles de memoria caché.

La memoria caché tiene las siguientes características:

- **L1I**: Tamaño de 16 KB, tiempo de acceso de 1ns y política escritura directa.
- **L1D**: Tamaño de 32 KB, tiempo de acceso de 1ns y política escritura directa.
- **L2**: Tamaño de 2 MB, tiempo de acceso de 5ns y política post-escritura.
- **Memoria principal**: Tamaño de 4 GB, tiempo de acceso de 100 ns.

El código ejecutado por cada núcleo se muestra a continuación. Ambos núcleos empiezan a ejecutar su código al mismo tiempo pero debido a los sleeps ejecutarán cada bucle en distintos instantes de tiempo. Las etiquetas T0, T1, T2, T3, T4 y T5 referencian distintos puntos de ejecución en cada programa. Observe que T0<T1<T2<T3<T4<T5.

Los índices de los bucles **i** y **j** y las variables **tmp1** y **tmp2** se almacenan en registros. El vector **a** se puede almacenar en caché pero **está inicialmente ubicado sólo en la memoria principal** (no está en la caché). Cada elemento de a ocupa una palabra. Asumir que el tiempo de ejecución de cada bucle es tan pequeño que es despreciable.

```
// Código ejecutado en núcleo 1
sleep(seconds(2));
// T1
for (i=0;i<40;++i) {
  tmp1 = tmp1 + a[i];
}
sleep(seconds(3));
// T2
for (j=0;j<40;++j) {
  a[j] = j;
}
// T3


// Código ejecutado en núcleo 2
// T0
for (i=0;i<40;++i) {
  tmp2 = tmp2 + a[i];
}
sleep(seconds(10));
// T4
for (j=0;j<40;++j) {
  tmp2 = tmp2 + a[j];
}
// T5
```

Se pide:

1. Describir en qué estados del protocolo **MSI** está el bloque que almacena **a[0]** para la caché L1D del **núcleo 1** en los instantes T0, T1, T2, T3, T4 y T5. Razona la respuesta.

	T0	T1	T2	T3	T4	T5
Estado de **a[0]** en L1D del núcleo 1						

2. Describir en qué estados del protocolo **MSI** está el bloque que almacena **a[0]** para la caché L1D del **núcleo 2** en los instantes T0, T1, T2, T3, T4 y T5. Razona la respuesta.

	T0	T1	T2	T3	T4	T5
Estado de **a[0]** en L1D del núcleo 1						

3. Calcular el tiempo de ejecución del código ejecutado por el **núcleo 2** considerando únicamente los tiempos de acceso a los datos (asumir que el tiempo de acceso a las instrucciones y el de ejecución de los bucles es despreciable).

Solution 6

Aparatado 1 El bloque **a[0]** está inicialmente en memoria. Durante la primera iteración del primer bucle, se accede a **a[0]** y se produce un fallo caché en L1D del núcleo 1 que se propaga a L2, produciendo un acierto. El bloque es transferido de la caché L2 a la L1D del núcleo 2. El estado del bloque pasa de *inválido* a *compartido* (I→S).

Posteriormente, tras 3 segundos, el núcleo 1 escribe en **a[0]** pasando de estado compartido a modificado (S→M) e invalidando la copia del núcleo 2. En t=5 segundos, el núcleo 2 lee **a[0]** pasando de estado modificado a compartido (M→S).

	T0	T1	T2	T3	T4	T5
Estado de **a[0]** en L1D del núcleo 1	I	S	S	I	I	S

Apartado 2 El bloque **a[0]** está inicialmente en memoria. Durante la primera iteración del primer bucle, se accede a **a[0]** y se produce un fallo caché en L1D del núcleo 2 que se propaga a L2 produciendo otro fallo. El bloque es transferido de memoria a la caché L2 y de allí a la L1D del núcleo 1. El estado del bloque pasa de *inválido* a *compartido* (I → S).

Posteriormente, en t=5 segundos, el núcleo 1 escribe en **a[0]** invalidando la copia del núcleo 2, pasando de *compartido* a *inválido* (S → I). En t=10 segundos, el núcleo 2 lee **a[0]** pasando de estado modificado a compartido (M → S).

	T0	T1	T2	T3	T4	T5
Estado de **a[0]** en L1D del núcleo 1	I	S	S	I	I	S

Apartado 3

- **Primer bucle** (índice **i**):
  - Cuando se ejecuta el primer bucle del núcleo 2 las entradas de a están en la memoria principal. El número de entradas por bloque es de $\frac{16}{4} = 4$ y el número de bloques accedidos es $\frac{40}{4} = 10$.
  - El tiempo de acceso para la primera entrada de cada bloque es el coste de acceder a memoria: $t = 1 + 5 + 100$ ns Para el resto de las entradas del bloque, dado que están en L1, el tiempo será el del acceso a la L1: $t = 1$ ns
  - El tiempo para cada bloque será: $T_{bloque} = 106 + 3 \cdot 1$ ns.
  - El tiempo del primer bucle será: $T_{bucle1} = 10 \cdot T_{bloque} = 1090$ ns.
- **Segundo bucle** (índice **i**):
  - Cuando se ejecuta el segundo bucle del núcleo 2 las entradas de a están ya en la caché L2. El número de entradas por bloque es de $\frac{16}{4} = 4$ y el número de bloques accedidos es $\frac{40}{4} = 10$.
  - El tiempo de acceso para la primera entrada de cada bloque es el coste de acceder a caché L2: $t = 1 + 5$ ns.
  - Para el resto de las entradas del bloque, dado que están en L1, el tiempo será el del acceso a la L1: $t = 1$ ns.
  - El tiempo para cada bloque será: $T_{bloque} = 6 + 3 \cdot 1$ ns
  - El tiempo del segundo bucle será: $T_{bucle2} = 10 \cdot T_{bloque} = 90$ ns.
- **Tiempo total**:
  - El tiempo total será: $T = T_{bucle1} + T_{bucle2} + sleep = 1180ns. + 10 * 10^9$ ns

Exercise 7 *Examen de enero de 2013.*

Dada la siguiente implementación de *lock/unlock*:

```
lock(location) {
  acquired = 0;
  while (! acquired) { // espera si lock ya está adquirido
    acquired = test_and_set(location);
      if (!acquired) // si test_and_set no adquiere el lock espera de 2 ms
        sleep (2ms);
  }
}

unlock(location) {
  location = 0;
}
```

Cuatro procesadores (P0,P1,P2 y P3) intentan adquirir el lock ejecutando simultáneamente el siguiente código:

```
lock(location);
// Realizar cómputo durante 3 ms
unlock(location);
```

Se pide:

1. Calcular cuánto tiempo (en ms) tarda en adquirir el cerrojo cada uno de los cuatro procesadores asumiendo que lo adquieren de forma ordenada (primero P0, luego P1, luego P2 y finalmente P3). Asumir que el tiempo de ejecución de cada sentencia y que la sobrecarga del protocolo de coherencia son despreciables.
2. ¿Es eficiente la implementación del cerrojo? Justifica tu respuesta.
3. Indicar una mejora posible para esta implementación. Justifica tu respuesta.

Solution 7

Apartado 1 Los tiempos que se tardan en adquirir son:

- P1: 0 ms
- P2: 4ms
- P3: 8ms
- P4: 12 ms

Apartado 2 No es eficiente porque el proceso que espera, espera más de lo necesario y se pierde tiempo (1ms) en esta espera.

Apartado 3 Alternativas más eficientes:

1. **test-and-set** con retardo exponencial.
2. Reducir el tiempo de espera en la versión actual (a costa de aumentar el tráfico de red),
3. Empleo de *load-linked/store-conditional*.
4. Empleo de un *mutex*.

Exercise 8 *Examen de junio de 2013.*

Considere 2 procesos ejecutando en sendos procesadores P1 y P2 en un multiprocesador de memoria compartida vía un bus de 100 Mbps. Los procesadores P1 y P2 disponen de una memoria caché no compartida inicialmente vacía y la consistencia se consigue utilizando el protocolo MSI. En el mismo instante de tiempo los procesos ejecutan el siguiente código, donde la variable **l** almacena la dirección de memoria de un cerrojo.

```
lock (l)
//cómputo por t=100 ms.
unlock(l)
```

Se pide:

1. Escribir la implementación para las primitivas de sincronización **lock(l)** y **unlock(l)** basada en la instrucción **test_and_set** cuya sintaxis es:

   test_and_set Registro Dir

2. Suponga que un fallo de caché implica una transferencia de 8 bytes para notificar el fallo y una invalidación implica una transferencia de 16 bytes. ¿Cuál sería el tiempo total de ejecución de este programa? Considere sólo los tiempos de fallo, invalidación y cómputo. Asuma que las cachés están inicialmente vacías.

3. ¿Qué diferencia de rendimiento existe entre las primitivas de sincronización **test_and_set** y **LL/SC** (*Load Linked* y *Store Condicional*)? Justifique su respuesta.

Solution 8

Apartado 1 La implementación usando la primitiva atómica *test-and-set* sería:

```
lock:   TST R1 direccion_l
        bnz R1 lock
        ret
unlock: sw 0 direccion_l
```

donde, la implementación de **TST** es equivalente a realizar de forma atómica:

```
TST:    lw R1 direccion_l #R1 se devuelve
        sw 1 direccion_l
        ret
```

Apartado 2 Se pide el tiempo de ejecución del programa, para lo cual calculamos inicialmente el coste en tiempo de fallo e invalidación. Un fallo de caché transfiere 8B sobre un bus de 100Mbps, por lo que el tiempo de transferencia sería:

$$t_{transf} = \frac{8 \cdot 8}{100Mbps} = \frac{64}{100} \cdot 10^{-6} = 0{,}64 \cdot 10^{-6} s$$

Una invalidación supone transferir 16B sobre un bus de 100Mbps:

$$t_{transf} = \frac{16 \cdot 8}{100Mbps} = \frac{128}{100} \cdot 10^{-6} = 1{,}28 \cdot 10^{-6} s$$

Supongamos que el proceso en P1 adquiere el cerrojo:

- **lock**: *test-and-set* R1 direccion_l → fallo + invalidación $(0{,}64 + 1{,}28) \cdot 10^{-6}$ s
- **cómputo**: 0.1s

- **unlock**: fallo + invalidación $(0{,}64 + 1{,}28) * 10^{-6}$ s.

El tiempo de ejecución del proceso 1 implica el tiempo de fallo de caché al leer la dirección del cerrojo dado que inicialmente la caché está vacía. Además implica una invalidación al P2 ya que el *test-and-set* escribe en la dirección de memoria. Posteriormente, el cómputo con una duración de 0.1 s y finalmente la instrucción **unlock()** que implica de nuevo fallo (puesto que se invalida la caché por el P2 que está realizando constantemente *test-and-set*) e invalidación, dado que se modifica también en el otro procesador.

$$Total_{P1} = 1{,}92 \cdot 10^{-6} + 0{,}1$$

El proceso 2 accede al lock una vez ha sido bloqueado por P1. Esto quiere decir que realizará espera activa haciendo *test-and-set* hasta que vea libre el cerrojo. La primera vez que el proceso invoca lock supone un fallo y una invalidación.

- **Primer lock**: *TST* R1 direccion_l → fallo + invalidación $\frac{0{,}64+1{,}28}{\cdot} 10^{-6}$ s
- Las subsiguientes veces que el proceso 2 invoca *lock*, no hay fallo ni invalidación dado que está en caché y en estado modificado. Una vez el proceso 1 hace *unlock*, el proceso 2 vuelve a realizar *test-and-set* esta vez con éxito, y supone 1 fallo y una invalidación.
- **Último lock**: *TST* R1 direccion_l → fallo + invalidación $\frac{0{,}64+1{,}28}{\cdot} 10^{-6}$ s
- **Cómputo**: = 0.1s
- **Unlock**: no hay ni fallo ni invalidación

$$Total_{P2} = 1{,}92 \cdot 10^{-6} + 0{,}1$$

$$Total = 1{,}92 \cdot 10^{-6} + 0{,}1 + 1{,}92 \cdot 10^{-6} + 0{,}1 = 3{,}84 \cdot 10^{-6} + 0{,}2$$

Apartado 3 Para buscar las diferencias en rendimiento, veamos la implementación de *ll-sc*:

```
lock:   ll R1 direccion_l
        bnez R1 lock
        sc 1 direccion_l error
        bnez error lock
        ret
```

La instrucción especial **sc** (*store conditional*) sólo escribe un 1 en la dirección del cerrojo si el valor previamente leído con **ll** en el registro **R1** (*load linked*) no ha sido modificada por ningún otro proceso. Por tanto, cuando se ejecuta el *lock* usando la implementación *ll-sc*, siempre se lee de la variable del cerrojo pero no siempre se escribe, causando un menor número de accesos a memoria e invalidaciones por escritura. Con *test-and-set*, cada llamada a lock, lee y escribe en la dirección del cerrojo.

2. Ejercicios complementarios de consistencia

Exercise 9

El siguiente programa se ejecuta sobre una máquina que dispone de 3 procesadores. Sus variables se encuentran en memoria compartida e iniciadas a **0**.

El procesador 1 ejecuta:

```
X=1;
```

El procesador 2 ejecuta:

```
Print(X);
Print(Y);
```

El procesador 3 ejecuta:

```
X=1;
Print(X);
```

Asumiendo que todos los accesos a las variables se ejecutan de forma atómica, conteste a las siguientes preguntas:

- Para cada uno de los siguientes resultados indique:
  - ¿Es posible que se obtenga el resultado?
  - Si el resultado es posible ofrezca un orden total de las instrucciones que produzcan el resultado.

Print(X) [P2]	**Print(Y)**	**Print(X)** [P3]
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

- Indique un orden global de instrucciones (y su correspondiente resultado) que sea imposible bajo consistencia secuencial, pero posible bajo otros modelos más relajados.

Solution 9

La solución a este ejercicio pasa por dibujar un grafo de dependencia en orden de programa y para cada caso añadir las relaciones de orden adicionales.

Caso 0,0,0:

```
Print X(2)
Print Y
Y=1
Print X(3)
X=1
```

Caso 0,0,1

```
Print X(2)
Print Y
Y=1
X=1
Print X(3)
```

Caso 0,1,0

```
Print X(2)
Y=1
Print Y
Print X(3)
X=1
```

Caso 0,1,1

```
Print X(2)
Y=1
Print Y
X=1
Print X(3)
```

Caso 1,0,0: No es posible
Caso 1,0,1:

```
X=1
Print X(2)
Print Y
Y=1
Print X(3)
```

Caso 1,1,0

```
Y=1
Print X(3)
X=1
Print X(2)
Print Y
```

Caso 1,1,1

```
X=1
Y=1
Print X(2)
Print Y
Print X(3)
```

Por tanto la única opción que no se produce con consistencia secuencial es la combinación 1,0,0. Sin embargo con consistencia relajada al eliminarse el requisito de orden de programa, se tiene:

```
Print Y
Y=1
Print X(2)
Print X(3)
X=1
```

Exercise 10

Repita el ejercicio 9 para el siguiente programa, con todas las variables inicialmente a cero.
Procesador 1:

```
A=1;
Print(flag);
```

Procesador 2:

```
while (flag==0) ;
Print(A);
```

Procesador 3:

```
flag=1;
Print(A);
```

Solution 10

La solución a este ejercicio pasa por dibujar un grafo de dependencia en orden de programa y para cada caso añadir las relaciones de orden adicionales.

Los eventos observables son:

- Print flag
- Print A(2)
- Print A(3)

Usando un grafo como el destino se tiene que si el procesador 1 imprime el valor flag como 0:

- Tiene que ocurrir antes que flag=1 (Procesador 3). Esto implica que el procesador 3 imprimirá un valor de 1.
- Una vez pasado el bucle while, se ha producido la asignación a 1 para flag, por lo que el procesador 2, también tiene que imprimir 1

Caso 1,0,0

```
flag=1
Print A(3)
while (flag==0)
Print A(2)
A=1
Print flag
```

Caso 1,0,1

```
flag=1
while (flag==0)
Print A(2)
A=1
Print A(3)
Print flag
```

Caso 1,1,0

```
flag=1
while (flag==0)
Print A(3)
A=1
Print A(2)
Print flag
```

Caso 1,1,1

```
flag=1
A=1
while (flag==0)
Print A(3)
Print A(2)
Print flag
```

Si se eliminan las restricciones de consistencia relajada, sería posible usar el siguiente orden:

```
Print A(2)
A=1
Print flag
while (flag==0)
flag=1
Print A(3)
```

Que se corresponde con la combinación A(2) → 0, flag → 0, A(3) → 1.

Exercise 11

Repita el ejercicio 9 para el siguiente programa, con todas las variables inicialmente a cero. Procesador 1:

```
A=1;
```

Procesador 2:

U=A;
B=1;

Procesador 3:

V=B;
W=A;

Solution 11

Se debe hacer un análisis para los posibles valores de **U**, **V** y **W**:
0,0,0

U=A
V=B
W=A
A=1
B=1

0,0,1

U=A
V=B
A=1
W=A
B=1

0,1,0

U=A
B=1
U=B
W=A
A=1

0,1,1

U=A
A=1
B=1
V=B
W=A

1,0,0

V=B
W=A
A=1
U=A
B=1

1,0,1

A=1
U=A
V=B
W=A
B=1

Ahora bien, si tanto **U** como **V** valen 1, esto significa que necesariamente la última instrucción en ejecutar debe ser **W=A**, y por tanto **A** también debe tomar el valor 1.

Por tanto es imposible que se dé el caso 1,1,0

Para que se diese tal caso, se debería eliminar la restricción de orden de programa como en la siguiente secuencia:

A=1
U=A
W=A
B=1
V=B

Exercise 12

El siguiente programa se ejecuta en una máquina con 2 procesadores. Sus variables están declaradas en memoria compartida e inicialmente valen 0.

Procesador 1:

X=1;
Y=X;

Procesador 2:

Y=2;
X=3;

- ¿Qué resultados para X e Y son posibles bajo consistencia secuencial?
- ¿Existe un resultado, que sea aceptable bajo consistencia relajada, pero no lo sea bajo consistencia secuencial?
- ¿Es el resultado (3,0) posible?

Solution 12

Por una parte, el valor final para **X**, solamente puede ser o 1 o 3.

Si el valor final de **X** es 1, la asingación **X**=3, debe preceder a **X**=1, y por tanto **Y** debe tomar también el valor 1.

Si el valor final de **X** es 3 se pueden dar los siguientes ordenes:

Caso 1 (X=3, Y=1)

Y=2
X=1
Y=X
X=3

Caso 2 (X=3, Y=3)

Y=2
X=1
X=3
Y=X

Caso 3 (X=3, Y=2)

X=1
Y=2
Y=X
X=3

Caso 4 (X=3, Y=2)

X=1
Y=X
Y=2
X=3

Caso 5 (X=3, Y=2)

X=1
Y=2
X=3
Y=X

El siguiente caso es posible bajo consistencia relajada:

Y=2
X=3
Y=X
X=1

Resultado (**X=1**, **Y=3**), que no es posible con consistencia secuencial.

Por último, el caso (3,0) no es posible ni sobre un modelo de consistencia relajada, porque **X=1** debe ejecutarse siempre antes que **Y=X**. Por tanto, es imposible que **Y** tome el valor 0.