Ejercicios Tema 3

① SMP, MESI, 4 nodos

Evento	N1	N2	N3	Acciones
Inicio	Inválido	Inválido	Inválido	-
Lectura N1	Exclusivo	Inválido	Inválido	N1 pide bloque, MP responde
Lectura N2	Compartido	Compartido	Inválido	N2 pide bloque, MP responde, N1 modifica estado
Escritura N1	Modificado	Inválido	Inválido	N1 y N2 modifican estado
Escritura N2	Inválido	Modificado	Inválido	N1 envía bloque e invalida, N2 recibe y modifica estado
Escritura N3	Inválido	Inválido	Modificado	N2 envía bloque e invalida, N3 recibe y modifica estado

② 8 nodos. 1 bit de estado en MP pr bloque, línea de caché de 64 byte,

Tamaño_directorio = Nº_bloques_memoria · Espacio-bloque-en-directorio

$$Nº\_bloques\_memoria = \frac{Tamaño\_Memoria\_Principal}{Tamaño\_Línea\_caché}$$

Espacio-bloque-en-directorio = 9 bits (1 bit presencia por nodo + 1 bit estado)

$$Tamaño\ directorio = \frac{TMP}{64B} \cdot 9b$$

$$\%\_de\_TMP = \frac{(TMP/64B)\cdot 9b}{TMP} \cdot 100 = \frac{9b}{64B \cdot 8b/B} \cdot 100 = 1{,}8\ \%$$

③ 1 bit de estado, 4 nodos, (8 GB cada uno), línea de caché de 64 Bytes

a) $Tamaño\ directorio = \frac{TMP}{TLC} \cdot EB = \frac{8GB \cdot 4}{64B} \cdot 5b = \frac{8 \cdot 10^9 \cdot 4 \cdot 8}{64 \cdot 8} \cdot 5 = 2{,}5\ GB$

Situación	Directorio	Cachés N1	N2	N3	N4	Acciones
Inicial	0 0 0 0	I	I	I	I	-
Lectura N1	1 0 0 0	C	I	I	I	N1 pide bloque, MP3 responde
Escritura N1	1 0 0 0	M	I	I	I	N1 pide escritura, MP3 responde (y se ignora)
Lectura N2	1 1 0 0	C	C	I	I	N2 pide, N1 responde
Lectura N3	1 1 1 0	C	C	C	I	N3 pide, N1 y N2 responden
Escritura N4	1 1 1 1	I	I	I	M	N4 pide, N1, N2 y N3 responden y modifican su estado.

④ a) Consistencia secuencial.

P1:	1	2	2
P2:	2	2	1

b) No se garantiza W → R (lectura puede adelantar a escritura previa)

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

(según donde se adelante las lecturas de los printf)

⑤ a) Consistencia secuencial

r1: 30
r2: 40

b) Consistencia de liberación (no se garantiza ningún orden)

r1 = 0 r2 = 0 (las lecturas se adelantan a las escrituras y a flag)
r1 = 0 r2 = 40 (solo se adelanta la lectura de x)
r1 = 30 r2 = 0 (" " " " " y)
r1 = 30 r2 = 40 (orden secuencial)

(6) Relaja $W \rightarrow R$

a) Siempre que pongamos la liberación del cerrojo tras la sección crítica se cumplen los requisitos del cerrojo

b) Con Itanium no se garantiza ningún orden, por lo que habría que utilizar las ~~xxxxxx~~ especiales que ofrece la arquitectura para implementar el cerrojo. (instrucciones)

(7) No se garantiza $W \rightarrow R$ ni $W \rightarrow W$

a = {1, 2, 3, 4, 5, 6, 7, 8} k = sum = 0 (compartidas) nthreads = 3

a) sum = 36 si los threads escriban secuencialmente

sum = 12
sum = 15
sum = 9
} si el último en escribir adelantó la lectura de sum al resto

sum = 27
sum = 21
sum = 24
} si se ignora una escritura de un thread

b) No se garantiza $W \rightarrow R$

La liberación del cerrojo no puede adelantar a las escrituras en sum porque se garantiza $W \rightarrow W$. Por tanto, se consigue la exclusión mutua y el resultado es 36

(8)
```
Barrera (id, num_procesos) {
  bandera_local = !(bandera_local)
  lock (bar[id].cerrojo);
    bar[id].cont++;
  unlock (bar[id].cerrojo);
  if (bar[id].cont == num_procesos) {
    bar[id].cont = 0;
    bar[id].bandera = bandera_local;
  }
  else
    while (bar[id].bandera != bandera_local) {};
}
```

Se sustituye cont_local por la variable compartida bar[id].cont.

Con esto se da el problema si un thread se bloquea justo antes de la asignación del if. Si el último en llegar a Barrera vuelve a poner la variable compartida a 0 el thread bloqueado (cuando continúe) se quedará indebidamente atrapado en la espera ocupada.

(9)
```
Barrera (id, num_procesos) {
  bandera_local = !(bandera_local);
  cont_local = FetchAndAdd (barr[id].cont, 1);
  if (cont_local = num_procesos){
    barr[id].cont = 0;
    barr[id].bandera = bandera_local;
  }
  else while (barr[id].bandera != bandera_local) {};
}
```

(10) No se garantiza W→R, paralelización dinámica

a) Fetch & O2

```
while (Fetch&O2 (k,1) == 1) {}
  i_local = i_compartida + 1;
k = 0;
```

(Funciona porque se garantiza W→W)

b) Fetch & O2 y Fetch&Add

```
i_local = Fetch&Add (i_compartida, 1);
```

(11) a) No se garantiza W→R, pero funciona correctamente porque se asegura W→W y la liberación de k=0 no puede adelantar al acceso a barr[id].cont

b) Con un modelo de consistencia de ordenación débil no se garantiza ningún orden, y por tanto no funciona correctamente. Esto ocurre si antes de escribir el valor actualizado de barr[id].cont un flujo lee el mismo valor, es decir, si se adelanta k=0 a la sección crítica.

Ejercicios Tema 3

Cuestiones

- Cuestión 1. Diferencias entre multithread temporal y multihread simultáneo.

Un multithread temporal ejecuta varios threads de forma **concurrente** en el mismo core, de forma que solo puede emitir instrucciones de un único thread en cada ciclo.

En cambio, un multithread simultáneo puede ejecutar varios threads **paralelamente**, de manera que en un ciclo puede emitir instrucciones de más de un thread.

- Cuestión 2. Diferencias entre multithread temporal de grano fino y de grano grueso.

En los de grano fino la conmutación de los threads se decide en cada ciclo y sin coste alguno, mediante técnicas como el RoundRobin. Por otro lado, los de grano grueso deciden la conmutación tras algún evento o intervalos de tiempo prefijados, ocasionando un coste de 0 o más ciclos para cambiar.

- Cuestión 3. Suponga un multiprocesador con protocolo MESI de espionaje. Si un controlador de cache observa en el bus un paquete de petición de lectura exclusiva de un bloque que tiene en estado C, debe (indique cuál sería la respuesta correcta y razone por qué es la respuesta correcta):
  a. Generar un paquete de respuesta con el bloque y pasar el bloque a estado I.
  b. Pasar el bloque a estado I.
  c. Generar un paquete de respuesta con el bloque y pasar el bloque a estado E.
  d. No tiene que hacer nada

La respuesta correcta es la b).

El controlador debe invalidar el bloque puesto que se trata de una petición de lectura exclusiva. No envía su bloque ya que la memoria lo tiene actualizado y esta se encargará de generar un paquete con él (por si el procesador que pide no tiene el bloque actualizado en caché)

- Cuestión 4. Suponga un multiprocesador con protocolo MESI de espionaje. Si un nodo observa en el bus un paquete de petición de lectura exclusiva de un bloque que tiene en estado M, ¿qué debe hacer? Razone su respuesta.

Debe invalidar el bloque y generar un paquete de respuesta con él, debido a que es la única copia actualizada en el sistema.

- Cuestión 5. Suponga un multiprocesador con el protocolo MESI de espionaje. Si el procesador de un nodo escribe en un bloque que tiene en su cache en estado I, debe (indique cuál sería la respuesta correcta y razone por qué es la respuesta correcta):
  a. Generar paquete de petición de acceso E al bloque y pasar el bloque a M
  b. Generar paquete de petición de acceso E y pasar el bloque a estado E
  c. Generar paquete de petición de acceso E con lectura y pasar el bloque a estado E
  d. Generar paquete de petición de acceso E al bloque con lectura y pasar el bloque a M

d) Ya que su copia no es válida debe previamente traer el bloque actualizado a su caché. Posteriormente (tras realizar la escritura) cambiará el estado a modificado pues será la única copia válida en el sistema

Ejercicios Tema 3

- Cuestión 6. ¿Cuál de los siguientes modelos de consistencia permite mejores tiempos de ejecución? Justifique su respuesta.
    a. modelo de ordenación débil
    b. modelo implementado en los procesadores de la línea x86
    c. modelo de consistencia secuencial
    d. modelo de consistencia de liberación

Siempre que se eviten todos los errores, el modelo de consistencia de liberación debería permitir los mejores tiempos de ejecución. Esto se debería a que el hardware no debe preocuparse de evitar las dependencias y puede reordenar el código de la forma más eficiente.

- Cuestión 7. Indique qué expresión no se corresponde con la serie (justifique su respuesta):
    a. Lock
    b. Fetch_and_Or
    c. Compare_and_Swap
    d. Test_and_Set

a) Lock no se corresponde con el resto de expresiones, ya que es una "instrucción" que se debe formar en base a las otras. Lock indica que se va a cerrar un cerrojo sobre una variable.