

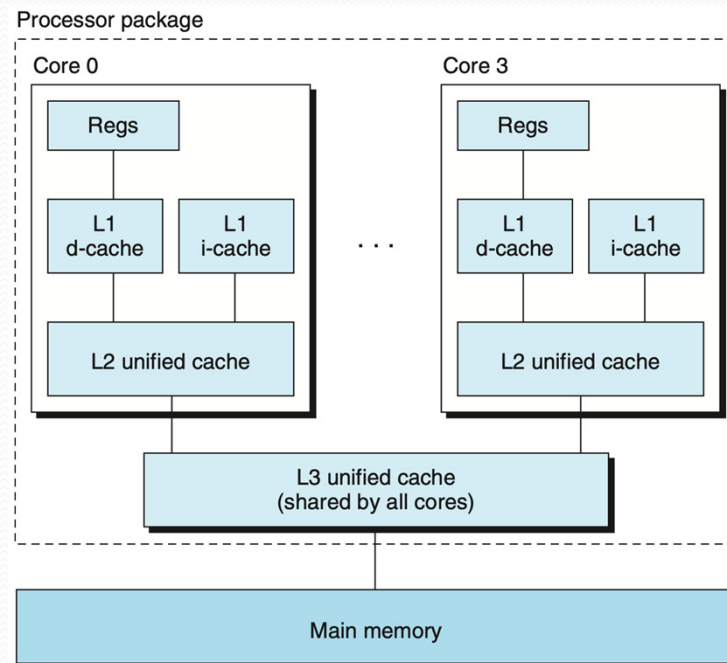
Coherencia de caché



Problema de la coherencia de caché

- Es posible que dos procesadores tengan en sus cachés dos valores distintos de la misma dirección.

Intel Core i7



Fuente: CS-APP, p. 668

Problema de la coherencia de caché

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

- Si B lee de nuevo el valor de X va a recibir 0.



Definición informal de coherencia

- Un sistema de memoria es coherente si cualquier lectura de un dato devuelve el valor escrito más reciente de ese dato.



Memoria compartida

- Hay dos aspectos críticos para escribir correctamente programas con memoria compartida.
- **Coherencia.** Qué valores pueden ser regresados por una lectura.
- **Consistencia.** Cuándo un valor escrito va a ser regresado por una lectura.

Coherencia

- Definición formal:
- P escribe X; P lee X (sin escrituras intermedias) la lectura devuelve el valor escrito.
- P1 escribe X; P2 lee X (suficientemente más tarde) la lectura devuelve el valor escrito por P1.
- P1 escribe X, P2 escribe X, todos los procesadores ven las escrituras en el mismo orden. Todos terminan con el mismo valor final para X.

Protocolo de invalidación de escritura

- Es un método para hacer cumplir la coherencia.
- Al ocurrir una escritura se invalidan todas las copias en los cachés.

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

Protocolo de invalidación de escritura

- ¿Qué sucede si dos procesadores intentan escribir el mismo dato simultáneamente?
- Uno de ellos gana y al otro se le invalida su copia.
- El otro procesador tiene que volver a leer el dato antes de escribirlo.
- A esto se le llama tener **acceso exclusivo** al dato.

Ejemplo

- Suponer lo siguiente:
- X vale 0 en la memoria.
- Las CPUs P1 y P2 tienen una copia de $X = 0$ en sus caches.
- P1 quiere incrementar a X en 1 y al mismo tiempo P2 quiere incrementar a X en 2.
- Si P2 gana, X se actualiza a 2 y se invalida la copia de X en P1.
- Esto obliga a P1 a volver a leer el valor de X y ahora lee $X = 2$.
- P1 actualiza X a 3 y ahora se invalida la copia de X en P2.

Problema de la consistencia

- ¿Cuánto tiempo debe pasar para que un valor escrito pueda ser regresado por una lectura?
- Para garantizar la consistencia, una escritura no se completa (ni se permite que ocurra la siguiente escritura) hasta que todos los procesadores hayan visto el efecto de esa escritura.
- Las lecturas se pueden reordenar, pero las escrituras se terminan en orden del programa.



Ejemplo en Java

- Hacer un programa que cree un hilo.

Solución

```
public static void main(String[] args)
{
    Runnable runnable = () -> {
        for (int i = 0; i < 10; i++) {
            System.out.println("Saludos " + i +
                " de: " + Thread.currentThread().getName());
        }
    };
    Thread thread = new Thread(runnable);
    thread.start();
}
```

Detener un hilo

- Suponer que se desea detener el hilo después de 1 segundo.
- El método `Thread.stop()` es obsoleto (deprecated) por ser inseguro.
- Ver <https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/doc-files/threadPrimitiveDeprecation.html>
- Una forma de detener un hilo es mediante una variable que sea revisada en el método `run()`.

Esto no debe funcionar

// Broken! - How long would you expect this program to run?

```
public class StopThread {  
    private static boolean stopRequested = false;  
    public static void main(String[] args) throws InterruptedException {  
        Runnable runnable = () -> {  
            int i = 0;  
            while (!stopRequested) {  
                i++;  
            }  
        }  
    }  
}
```


Esto no debe funcionar

```
Thread thread = new Thread(runnable);  
thread.start();  
TimeUnit.SECONDS.sleep(1);    // 1 second  
stopRequested = true;  
}  
}
```


Explicación

- Explicación de *Effective Java, 3rd edition* por Joshua Bloch, p. 313.
- El problema es que, en ausencia de sincronización, no hay garantía de cuándo, si es que alguna vez, el hilo en segundo plano verá el cambio en el valor de `stopRequested` realizado por el hilo principal.
- Una solución es usar el keyword `volatile`.

Keyword volatile

- El keyword `volatile` garantiza que cualquier hilo que lea una variable verá el último valor escrito.
- `volatile` asegura dos cosas:
 1. Cada vez que se actualice la variable, se escribe en la memoria principal.
 2. Cada vez que se lea la variable, se lee de la memoria principal.
- Si solo un hilo actualiza la variable y los demás solo leen el valor, `volatile` es suficiente.

Solución con volatile

// Cooperative thread termination with a volatile field

```
public class StopThread {  
    private static volatile boolean stopRequested;  
    public static void main(String[] args) throws InterruptedException {  
        Runnable runnable = () -> {  
            int i = 0;  
            while (!stopRequested) {  
                i++;  
            }  
        }  
    }  
};
```


Solución con volatile

```
Thread thread = new Thread(runnable);  
thread.start();  
TimeUnit.SECONDS.sleep(1);    // 1 second  
stopRequested = true;  
}  
}
```