

Proyecto 1 - Protocolos de coherencia en sistemas multiprocesador

Juan Daniel Esquivel Rojas
juanesro1012@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

Abstract—The following report will explain the design and implementation of a simulation of a coherence protocol in multiprocessor systems. Besides, the report explains the tools used for development such as the write policy and the MSI coherence protocol.

Palabras clave—Arquitectura, Cache, Computadores, Memoria, MSI, Multiprocesador, Política de escritura, Protocolo de Coherencia

I. INTRODUCCIÓN

Los sistemas multiprocesador corresponden a la base funcional de los computadores modernos de la actualidad. Se pueden encontrar en cualquier sistema como los chip, sistemas embebidos y conjuntos de servidores los cuales desempeñan una función dentro de la computación de alto desempeño y procesamiento de gran cantidad de datos en paralelo. Una de las mayores complicaciones con la que cuentan estos sistemas es que al poseer múltiples procesadores su comunicación entre los mismos se dificulta. En la actualidad investigadores y especialistas de estos sistemas buscan mecanismos para optimizar la capacidad para compartir tareas y recursos entre los diferentes procesadores. La memoria se comparte con los procesadores y corresponde a uno de los componentes más importantes, puesto a que se compone de los datos necesarios para ejecutar las instrucciones. El tener una memoria compartida representa un desafío muy grande, ya que múltiples procesadores pueden solicitar un mismo dato en un instante de tiempo, por lo que se generan inconsistencias en los datos, esto es lo que se conoce como incoherencia de caché. Este problema en la actualidad aun se investiga y se busca una solución dentro del área de arquitectura de computadores. En este caso, para este proyecto se debe de implementar una simulación de un sistema multiprocesador con una memoria compartida a través de dos niveles de caché: L1 para cada núcleo dentro del sistema y L2 compartida entre los mismos núcleos. Para el desarrollo de este proyecto se debe utilizar los conceptos de caché como lo es los protocolos de coherencia basados en monitoreo y directorios, además de las políticas de escrituras.

II. DESARROLLO DEL SISTEMA

II-A. Descripción de la solución

Según el problema de la coherencia de caché en los sistemas multiprocesadores, se diseña una simulación utilizando el

lenguaje de programación Java, en el cual se diseñan los procesadores a utilizar, los niveles de la memoria de cache y la memoria principal de un sistema multiprocesador en el cual se utiliza como política de escritura Write-Through y se utilizan protocolos de coherencia basados en snooping (monitoreo) y en directorios y el MSI. Se escogió dicha política de escritura ya que al diseñar la simulación usando el paradigma de programación orientada a objetos que ofrece Java es mucho más fácil que el implementar Write-Back, además de que no se necesita implementar Hardware, lo cual es favorable para no tomar en cuenta aspectos de rendimiento y tiempo real.

II-B. Implementación del Diseño

II-B1. Protocolo de coherencia: Para la implementación de la coherencia de caché del sistema a desarrollar, se dará uso al protocolo MSI, el cual cuenta con menos estados que otros protocolos como el MESI y MOESI, y por lo tanto es ventajoso a la hora de desarrollar y programar el sistema, ya que al ser una simulación de un sistema multiprocesador no es necesario tomar en cuenta más estados. Los estados del protocolo se detallan a continuación:

- **Modificado:** La dirección objetivo está en la memoria caché, pero su contenido está modificado y no es coherente con la memoria del sistema. Ninguna otra caché en el sistema posee esos datos.
- **Compartido:** La dirección objetivo está en la memoria caché y también está en al menos otra unidad más. Los datos son coherentes con la memoria del sistema.
- **Inválido:** Sucede cuando la dirección objetivo no está en la memoria caché.

Teniendo esto en cuenta, se diseña un diagrama de estados del protocolo para el sistema multiprocesador a simular.

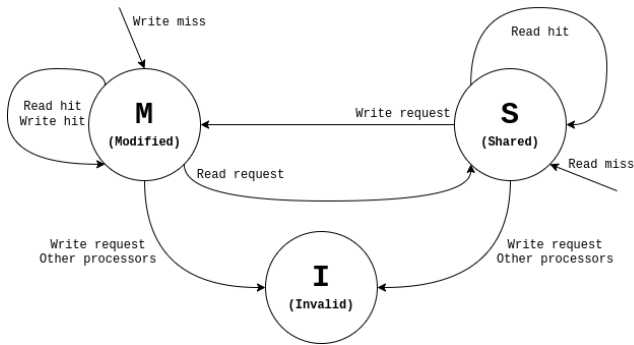


Figura 1: Diagrama de estados del protocolo de coherencia MSI.

II-B2. Diagrama de bloques del sistema: Como parte del sistema multiprocesador a desarrollar, se crea un diagrama de bloques de alto nivel (Fig. 2), en el cual se muestran las diversas unidades a desarrollar y su conexión entre las mismas para su correcto funcionamiento. El sistema cuenta con dos chips los cuales poseen dos procesadores cada uno, con su respectiva memoria caché L1 y con otro nivel de memoria caché denotada como L2, la cual es accesible por ambos procesadores. Esta memoria L2 es compartida, por lo que es la misma para ambos chips. La comunicación entre los chips es creada a base de flags, al igual que la comunicación de los procesadores con las memorias caché y principal.

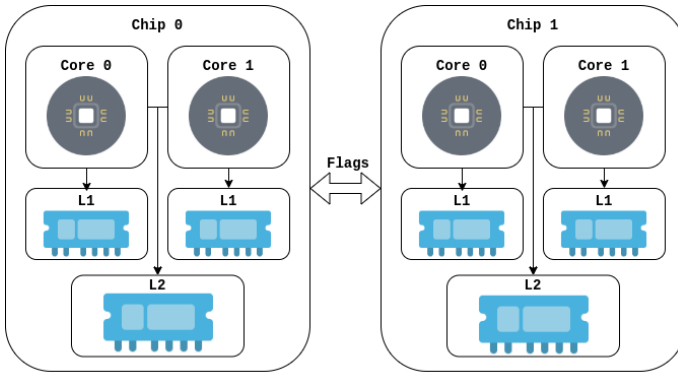


Figura 2: Diagrama de bloques del sistema multiprocesador.

II-B3. Política de escritura implementada: Según los diagramas del protocolo de coherencia MSI y la política de escritura escogida (Write-Through), se crea un diagrama de flujo, siguiendo paso a paso el funcionamiento del sistema. Dicho diagrama se presenta en la siguiente figura 3.

Al crearse una nueva instrucción inicia el sistema a ejecutarse, verifica el contenido de la instrucción y si es de escritura, se escribe en todas las memorias, según la política de escritura establecida. Si la operación es de lectura, se lleva a cabo el flujo de buscar y leer el dato por todas las memorias, es decir, primero se busca el dato de L1, si se encuentra es leído y se termina el trabajo, pero si no, se repite el flujo pero en el siguiente nivel de caché (L2), y si no se encuentra, se termina yendo a buscar el dato a la memoria principal.

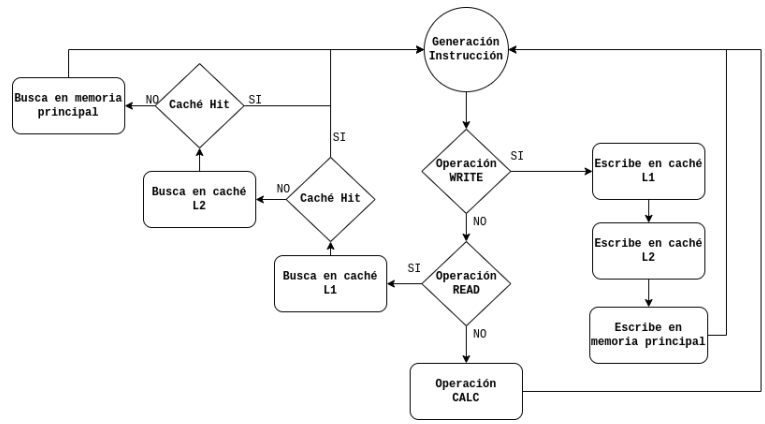


Figura 3: Diagrama de flujo del funcionamiento del sistema.

II-B4. Software del sistema: Teniendo en cuenta la política de escritura, y el protocolo de coherencia a implementar en el sistema, se desarrolla el mismo bajo programación orientada a objetos en Java, utilizando las bibliotecas de Thread, y Log para que el sistema cumpla con su función básica. La creación de las clases está dada según el diagrama de bloques del sistema, y con ello se crea el diagrama de clases de la solución al desarrollo del sistema multiprocesador.

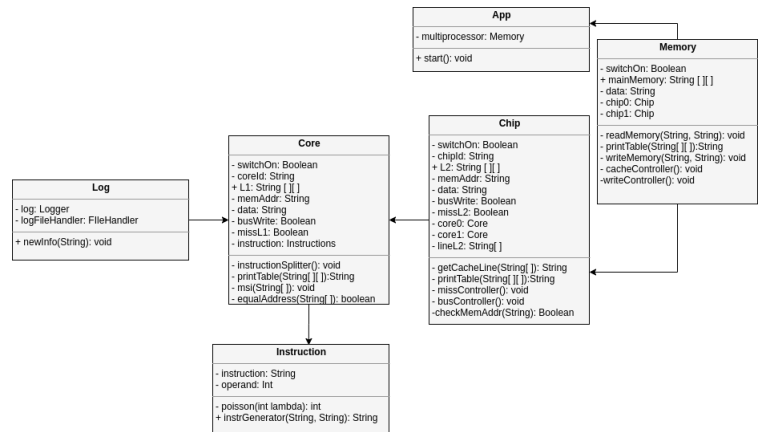


Figura 4: Diagrama de clases del sistema.

III. RESULTADOS

El sistema, al no contar con una interfaz gráfica que evidencie el constante cambio tanto en las memorias caché como en la memoria principal, se implementa una serie de archivos .log, los cuales guardan todos los eventos producidos dentro del sistema al entrar en ejecución. Estos archivos también guardan el estado de las memorias caché y la memoria principal siempre que se acceden o que modifican valores en las mismas. A continuación se adjuntan los resultados obtenidos de la simulación del sistema desarrollado, dentro de los archivos .log, generados para cada procesador dentro del sistema multiprocesador, y en la terminal de comandos vista desde el programa VisualStudio Code.

```

OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

INFO: Cache L1 de: P0-0
Block||Coherence||Mem.Address||Data
0||I||null||null
1||M||0011||8a89
New instruction!-----
P0,0,READ,1101 to: P0-0
Jun 02, 2020 7:12:56 PM app.Log newInfo
INFO: New instruction: P0,0,READ,1101
CALC operation w success
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: CALC operation w success
Instruction done!-----
Cache L1 de: P0-1

```

Figura 5: Ejecución del sistema desarrollado.

El funcionamiento sincrónico de los procesadores en ejecución genera que dependiendo a la instrucciones que ejecute, uno va completar más en un tiempo determinado que otro, por ello en la figura 5 se puede notar como se traslapan los eventos de los distintos procesadores en x tiempo.

```

Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: Prueba
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: Cache L1 de: P0-0
Block||Coherence||Mem.Address||Data
0||I||null||null
1||I||null||null
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: New instruction: P0,0,WRITE,0011,8a89
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: WRITE operation on L1: P0 - 0
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: Main Memory:
Block||Owner||Data
0000||null||0000
0001||null||0000
0010||null||0000
0011||P0||8a89
0100||null||0000
0101||null||0000
0110||null||0000
0111||null||0000
1000||null||0000
1001||null||0000
1010||null||0000
1011||null||0000
1100||null||0000
1101||null||0000
1110||null||0000
1111||null||0000

Jun 02, 2020 7:12:56 PM app.Log newInfo
INFO: WRITE operation on L1: P0 - 0: SUCCESS

```

Figura 6: Archivo .log de la unidad P0-0.

En la anterior figura se evidencia el archivo .log del procesador P0-0, el cual muestra su memoria L1 (los valores con null para una mayor estética de la vista), ejecuta una instrucción WRITE y se evidencia que la escritura del dato en memoria principal fue concretada y su finalización con éxito.

```

P0-0.log x  P0-1.log x
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: Prueba
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: Cache L1 de: P0-1
Block||Coherence||Mem.Address||Data
0||I||null||null
1||I||null||null
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: New instruction: P0,1,CALC
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: Main Memory:
Block||Owner||Data
0000||null||0000
0001||null||0000
0010||null||0000
0011||P0||8a89
0100||null||0000
0101||null||0000
0110||null||0000
0111||null||0000
1000||null||0000
1001||null||0000
1010||null||0000
1011||null||0000
1100||null||0000
1101||null||0000
1110||null||0000
1111||null||0000

Jun 02, 2020 7:12:56 PM app.Log newInfo
INFO: CALC operation: P0 - 1
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: CALC operation w success

```

Figura 7: Archivo .log de la unidad P0-1.

En la anterior figura se muestra el archivo del procesador P0-1, en el cual se ejecuta una instrucción CALC, la cual se ejecuta con éxito. Además en comparación con el anterior archivo .log mostrado, se puede ver que la memoria principal está actualizada con el cambio realizado por el procesador P0-0 anteriormente.

```

P0-0.log x  P0-1.log x  P1-0.log x
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: New instruction: P1,0,READ,1010
Jun 02, 2020 7:12:47 PM app.Log newInfo
INFO: Main Memory:
Block||Owner||Data
0000||null||0000
0001||null||0000
0010||null||0000
0011||P0||8a89
0100||null||0000
0101||null||0000
0110||null||0000
0111||null||0000
1000||null||0000
1001||null||0000
1010||null||0000
1011||null||0000
1100||null||0000
1101||null||0000
1110||null||0000
1111||null||0000

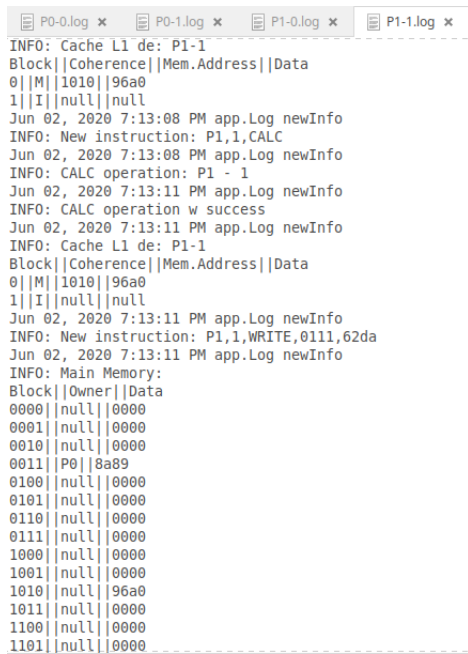
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: READ operation on L1: P1 - 0
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: No data on L1: P1 - 0
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: Search data on L2 from: P1 - 0
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: Search data on L2 from: P1 - 0
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: L2: Data from: P1 - 0 not found
Jun 02, 2020 7:12:59 PM app.Log newInfo
INFO: Another L2: Search data from: P1 - 0

```

Figura 8: Archivo .log de la unidad P1-0.

Según el archivo .log del procesador P1-0, se puede ver parte de la ejecución de una instrucción READ, en la cual, al no tener la dirección buscada en L1, procede a buscar la dirección en la caché L2, la cual es compartida por ambos

chips. Además se vuelve a evidenciar que la memoria principal es la misma para cada procesador.



```

P0-0.log x P0-1.log x P1-0.log x P1-1.log x
INFO: Cache L1 de: P1-1
Block|Coherence|Mem.Address|Data
0|M|1010|96a0
1|I|null|null
Jun 02, 2020 7:13:08 PM app.Log newInfo
INFO: New instruction: P1,1,CALC
Jun 02, 2020 7:13:08 PM app.Log newInfo
INFO: CALC operation: P1 - 1
Jun 02, 2020 7:13:11 PM app.Log newInfo
INFO: CALC operation w success
Jun 02, 2020 7:13:11 PM app.Log newInfo
INFO: Cache L1 de: P1-1
Block|Coherence|Mem.Address|Data
0|M|1010|96a0
1|I|null|null
Jun 02, 2020 7:13:11 PM app.Log newInfo
INFO: New instruction: P1,1,WRITE,0111,62da
Jun 02, 2020 7:13:11 PM app.Log newInfo
INFO: Main Memory:
Block|Owner|Data
0000|null|0000
0001|null|0000
0010|null|0000
0011|P0|8a89
0100|null|0000
0101|null|0000
0110|null|0000
0111|null|0000
1000|null|0000
1001|null|0000
1010|null|96a0
1011|null|0000
1100|null|0000
1101|null|0000

```

Figura 9: Archivo .log de la unidad P1-1.

Finalmente, para el archivo .log del procesador P1-1, se puede observar como luego de una operación WRITE se actualiza su memoria caché L1 con el dato y la dirección procesadas, al igual como se actualiza la memoria principal, la cual también tiene escrito el mismo dato, evidenciando así el funcionamiento del Write-Through. A su vez se expone un problema de implementación encontrado en el sistema. El cual es que no se actualiza ni se cambia el valor de lo que es la columna owner de la memoria principal, ya que se actualiza con el primer valor escrito en memoria, pero no con los demás.

IV. CONCLUSIONES

Los protocolos de coherencia de caché son de vital importancia para el desarrollo de los computadores en la actualidad, ya que si se desea un funcionamiento eficiente y rápido del sistema, se debe de tener bajo control lo que es la incoherencia de las memorias dentro del sistema.

En sistemas multiprocesadores y de trabajo a gran escala, es necesario implementar técnicas y protocolos para mantener la coherencia del caché y de las demás memorias entre todos los procesadores que forman el sistema.

Diseñar e implementar sistemas de simulación de coherencia de caché, genera un mayor enfoque educativo para entender mejor los diferentes protocolos de coherencia de caché y el cómo implementarlos dependiendo al problema o sistema específico que se requiera elaborar.

V. REFERENCIAS BIBLIOGRÁFICAS

REFERENCIAS

[1] Patterson, D., Hennessey, J., "Computer Organization and Design (4th ed.)". Morgan Kaufmann, 2009.

- [2] Hennessey J. L., Patterson, D. A., "Computer Architecture: A Quantitative Approach".
- [3] Hwang, K., Xu, Z., "Scalable Parallel Computing: Technology, Architecture, Programming". McGraw-Hill, New York, NY, 1998. ISBN 0-07-031798-4.
- [4] Herlihy, M., Shavit, N., "The Art of Multiprocessor Programming", Elsevier.