Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
CE-4302 Arquitectura de Computadores II

Proyecto I
Documento de diseño

Constituido por:

Alejandra Castrillo Muñoz - 2015155759

Profesor:
Ing. Luis Barboza Artavia

Cartago
I Semestre 2020

Índice

1. Requerimientos

Se puede considerar el siguiente texto como la descripción general del sistema:

> Aplicación de software que realice un modelo de un sistema multiprocesador con dos procesadores (chip) que a su vez tienen dos elementos de procesamiento (núcleo). Estos componentes se encuentran conectados a una memoria compartida, a través de una una caché L2 para cada chip y una caché L1 para cada núcleo. Ambas memorias caché son mapeadas directamente. Cada núcleo deberá generar solicitudes de procesamiento o acceso a memoria (lectura o escritura) a diferentes regiones de memoria. Sobre las memorias caché se deberá implementar el protocolo basado en directorios para solucionar el problema de las incoherencias en regiones compartidas de memoria.

Con esta descripción y los detalles del sistema en la especificación se generan los requerimientos tomando en cuenta la individualidad del elemento en cuestión y sus distintas necesidades de implementación. Para la consideración de los requerimientos se tomará en cuenta las posibilidades de lenguaje de desarrollo python y c.

1.1. Componentes y estructura del sistema

Para un mejor entendimiento de los componentes del sistema multiprocesador a desarrollar se puede tomar en cuenta el diagrama modular mostrado en la figura 1.

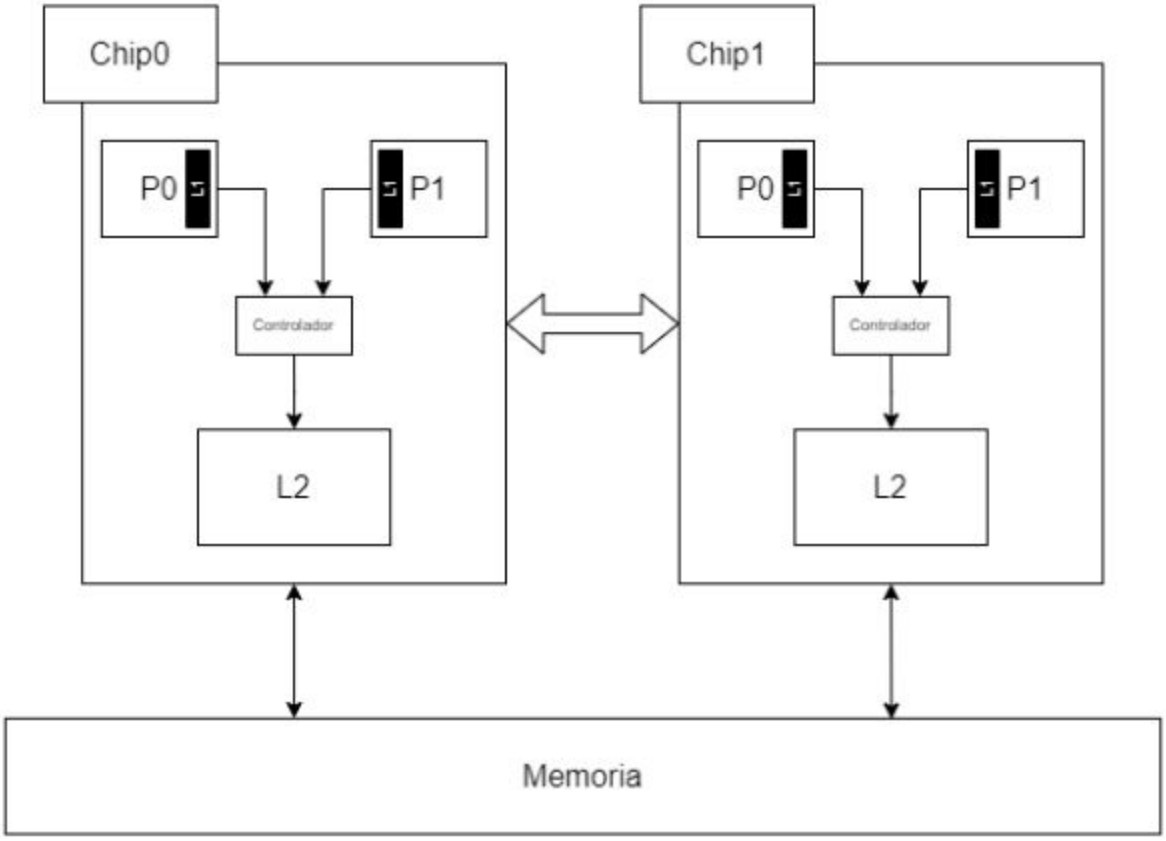

Figura 1. Arquitectura general del sistema

De este diagrama se puede obtener los componentes básicos:

- Procesador: se encarga de ejecutar las instrucciones del programa.
- Memoria caché L1: memoria de primer nivel que va incluida en el procesador. Ofrece el menor tiempo de respuesta en comparación a las otras memorias.

P0, 0

	Estado de coherencia	Dirección de memoria	Dato
0	M	1011	FFFF
1	S	0110	FFFF

Figura 2. Modelo de memoria caché L1

- Memoria caché L2: memoria de segundo nivel compartida por los procesadores dentro del chip, comúnmente más lenta que la memoria caché L1.

L2, 0

	Estado	Dueño	Dirección de memoria	Dato
0	DM	P0,1	1011	FFFF
1	DS	P0,0;E	0110	FFFF
2	DI	P1,0	0111	FFFF
3	DS	P1,0	1000	FFFF

Figura 3. Modelo de memoria caché L2

- Memoria principal: almacenamiento a largo plazo de los datos, tiene mayor tamaño por lo que su latencia es mucho mayor a las otras memorias.

M

	Estado	Dueño(s)	Dato
1011	DM	C0	FFFF
0110	DS	C0, C1	FFFF
0111	DI	C1	FFFF
...	...	...	...
1000	DS	C0	FFFF

Figura 3. Modelo de memoria principal

- Controlador: se encarga de manejar el protocolo de coherencia entre las caché L1 y L2. Maneja las transacciones de protocolo y la comunicación entre las caché.

Estos corresponden a componentes generales de los cuales se deben generar instancias, por lo que para su desarrollo es conveniente considerar lenguajes de programación con soporte de programación orientada a objetos. Además, debe haber comunicación entre los componentes para la implementación del sistema en conjunto.

1.2. Paralelismo

Como se mostró en la sección 1.1 de componentes, estos son instancias de un mismo objeto que deben ejecutarse de forma simultánea, por lo que se necesita alguna herramienta que provea ejecución paralela como los hilos. Se debe generar un hilo por cada instancia para simular el sistema multiprocesador.

En el caso de python existe la biblioteca Threading la cual viene incluida con el lenguaje. Para C existe Pthread, el cual es un modelo de ejecución paralela que permite la generación de hilos. Ambas bibliotecas proveen lo necesario para la ejecución de los componentes de forma paralela.

1.3. Correspondencia

Se debe implementar una correspondencia directa tanto en caché L1 como en L2. Este es un tipo de correspondencia en el que cada bloque de memoria está mapeado a un bloque en específico de caché. El bloque de caché al que se mapea el de memoria viene dado por la siguiente fórmula:

$$B_{cache} = B_{mem} mod\ NB_{cache}$$

En donde NBcache sería la cantidad de bloques que contiene la caché.

1.4. Protocolo MSI

Se debe utilizar el protocolo MSI como protocolo de coherencia de caché para las memorias caché L1. El protocolo MSI usa los tres estados necesarios en cualquier caché post-escritura para distinguir bloques válidos que no han sido modificados ("clean") de aquellos que han sido modificados ("dirty"). El estado de **inválido** tiene un significado claro. **Compartido** significa que el bloque está presente en la caché y no ha sido modificado, la memoria principal está actualizada y cero o más caché adicionales pueden tener también una copia actualizada (compartida). **Modificado** significa que únicamente este procesador tiene una copia válida del bloque en su caché, la copia en la memoria principal está anticuada y ninguna otra caché puede tener una copia válida del bloque (ni en estado modificado ni compartido).

Dentro de este protocolo se toman en cuenta las siguientes transacciones:

- **PrRd**: solicitud del procesador para lectura de una línea de caché.
- **PrWr**: solicitud del procesador para escritura de una línea de caché.
- **BusRd**: en caso de un read miss, se solicita al bus por el bloque de caché.

- **BusRdX**: en caso de que haya un write miss, se solicita al bus por el bloque de caché, e invalida el bloque en las demás caché.
- **BusUpgr**: en caso de un write hit, se le solicita el bus invalidar el bloque en las otras caché.
- **Flush**: el bloque de caché se ha escrito en memoria.

Estas transacciones definen el estado de los bloques de la memoria. El diagrama de estados del protocolo se puede observar en la figura .

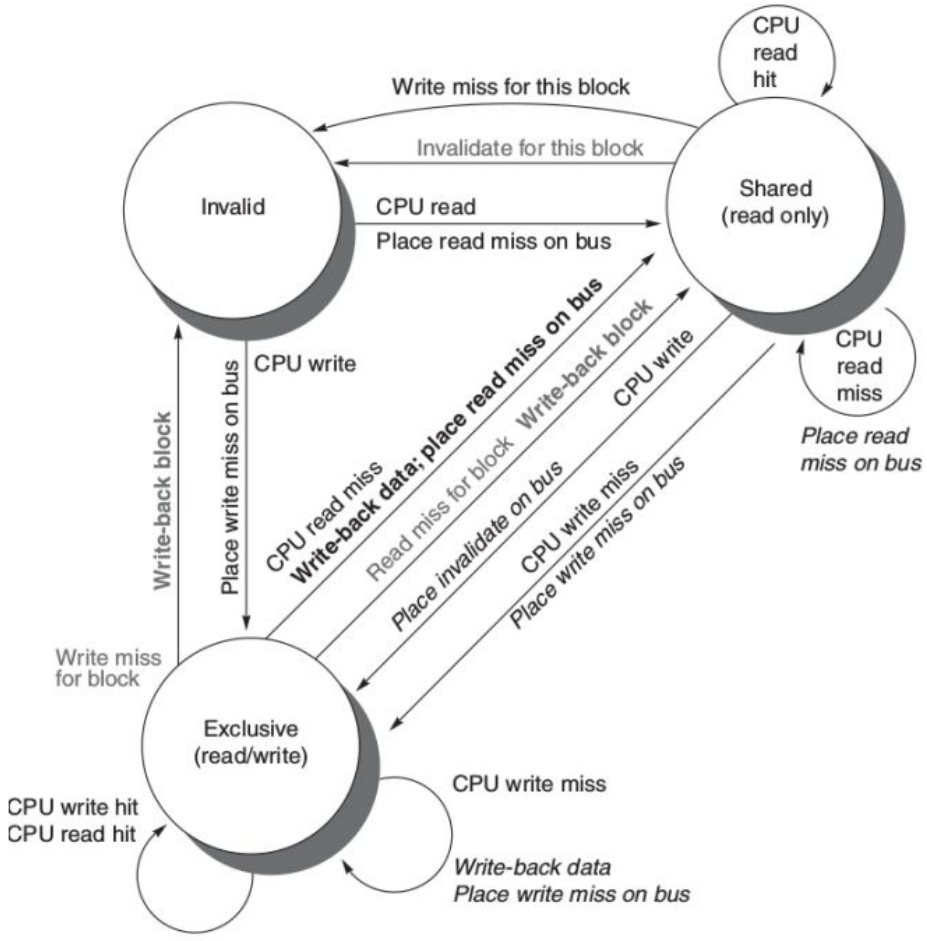

Figura 4. Máquina de estados del protocolo MSI

Para la implementación de este protocolo se debe utilizar un controlador que maneja la máquina de estados finitos mostrada en la imagen anterior. El controlador se encargará de generar y enviar las solicitudes entre las caché. Como parte de esto está generar la invalidación por escritura para mantener la coherencia.

1.5. Protocolo basado en directorios

Se debe implementar un protocolo basado en directorios para la memoria caché L2 y la memoria principal. En este caso no se tiene un directorio aparte a la memoria para llevar el control de la coherencia, sino que la información del directorio debe ser introducida dentro de las memorias como se mostró en la sección 1.1. Tanto L2 como la memoria llevan control de los dueños y el estado de cada bloque asociado a las direcciones de memoria.

El protocolo corresponde uno distribuido ya que al estar implementado dentro de la memoria, cada una de estas cuenta con su propio directorio incluido.

Al igual que en la memoria caché L1 se manejan 3 estados:

- DI: directorio inválido
- DS: directorio compartido
- DM: directorio modificado.

Además se incluye el listado de los dueños del bloque y si es compartido por otro chip. Tomando el estado DI como uncached y el DM como exclusivo, se puede considerar la siguiente máquina de estados para el desarrollo del protocolo.

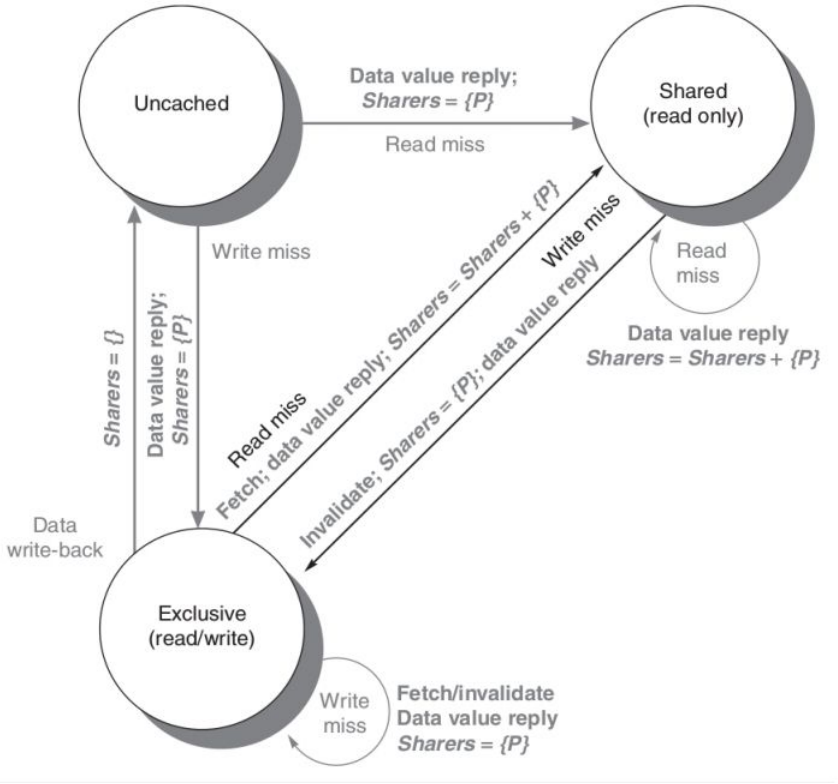

Figura 5. Máquina de estados del protocolo basado en directorios

1.6. Política de escritura

En este caso debido al objetivo del proyecto y a los protocolos implementados la política de escritura a utilizar es write-back, para poder visualizar de mejor manera la escritura y el progreso de las transacciones de los protocolos con escritura de punto a punto.

1.7. Tiempos de ejecución

Se debe asignar una frecuencia al sistema de manera que se pueda simular el tiempo de acceso que requiere cada tipo de memoria, lo que varía los tiempos de ejecución de las instrucciones considerablemente.

Un ejemplo de esto se puede ver en la reseña del análisis de rendimiento “Performance Analysis Guide for Intel® Core™ i7 Processor and Intel® Xeon™ 5500 processors” en este se obtienen las siguientes latencias asociadas a las memorias:

- Local L1 CACHE hit, ~4 cycles (2.1 - 1.2 ns)
- Local L2 CACHE hit, ~10 cycles (5.3 - 3.0 ns)
- Local DRAM ~60 ns

1.8. Log del sistema

Se debe implementar un log para el sistema en donde se puedan visualizar los siguientes eventos:

- Ejecución de una instrucción: mostrar cuál instrucción y quién la está ejecutando.
- Transacciones de los protocolos de coherencia de caché.
- Caché hit.
- Caché miss.
- Accesos a memoria.

1.9. Visualización de memorias

Se debe mantener una visualización constante de las memorias del sistema. Cada memoria puede verse como una matriz de datos lo que facilita su visualización por medio de escrituras en consola, sin embargo, se deben poder observar todas al mismo tiempo para poder mostrar los cambios en cada una a raíz de las instrucciones que realiza y las acciones de los protocolos, por lo que se debe considerar una interfaz gráfica que permita su visualización constante y simultánea.

Python contiene la biblioteca Tkinter, la cual ofrece un set de herramientas para el desarrollo de interfaces de usuario. Esta es sumamente sencilla de implementar y cuenta con los componentes necesarios para la visualización de tablas.

1.10. Generación de instrucciones

Se toman en cuenta 3 tipos de instrucciones: CALC, READ y WRITE. Cuál instrucción se debe generar quedará determinado por una distribución de probabilidad formal (binaria, de Poisson, hipergeométrica, etc). Existe un biblioteca llamada numpy de Python que provee esta fórmula de distribución, en el caso de C habría que generar la fórmula.

El encabezado de la instrucción debe ser el tipo al que esta corresponde. Las instrucciones CALC no necesitan de otros datos. A las instrucciones READ, se les debe agregar la dirección que desea leer y a las instrucciones WRITE se le debe proveer la dirección y el dato que quiere almacenar en ella.

2. Propuestas

2.1. Propuesta I

2.1.1. Lenguaje de programación

Se propone el lenguaje de programación C, ya que provee todas las herramientas necesarias para la implementación del sistema, con las cuales trabajé hace poco por lo que facilita la etapa de implementación. Por otro lado la interfaz de usuario se realizaría por medio de el lenguaje Python con la biblioteca Tkinter para facilitar el desarrollo.

2.1.2. Visualización

Como ya se mencionó en esta parte se utiliza el lenguaje de programación python, este tiene la posibilidad de utilizar programas en C como bibliotecas por lo que se realizaría la integración de ambos por este medio. Esto para poder utilizar Tkinter para el

desarrollo de la interfaz de usuario donde se muestren las tablas correspondientes al sistema multiprocesador.

2.1.3. Paralelismo

El lenguaje C provee múltiples bibliotecas para la implementación de hilos, la más conocida y utilizada corresponde a Pthreads, la cuál además he utilizado recientemente por lo que facilita su implementación.

2.1.4. Generación de instrucciones

Se propone una distribución binomial, con esta se puede obtener el tipo de función tomando evaluando cual da como resultado positivo según su probabilidad establecida. En este caso se tiene la opción de la implementación de una función que provea la distribución, esta tiene una implementación sencilla y ha sido previamente desarrollada por lo que su integración en el sistema no sería un problema.

2.1.5. Tiempos de ejecución

Para la latencia se propone la implementación de un reloj (clock) general que lleve el avance de los ciclos de forma que se pueda agregar el costo en CPI de cada ejecución o acceso a memoria tal como ocurriría en el sistema real. Se utilizaría como referencia el caso de el sistema Core i7 Xeon mostrado en la sección 1.7, el acceso a las caché se considera despreciable ya que de otra forma habría que considerar el costo por ciclo muy bajo para que el costo por acceso a memoria sea considerable y notorio para el usuario.

2.1.6. Log del sistema

Para implementar el log C no cuenta con instrucciones o bibliotecas incluidas con este fin por lo que se propone la creación de un archivo .c que cumpla con este fin. Este contendría todas las funciones necesarias para el manejo del log y sería accesible por todos los archivos del programa.

2.1.7. Componentes del sistema

Cada componente contaría con su propio header y source, en donde se implementa la funcionalidad de cada uno y structs para realizar las múltiples instancias requeridas de cada componente.

2.1.8. Protocolos

Para la implementación de los protocolos es necesaria la implementación de buses que se agregarían por medio de un struct con variables por las cuales los diversos componentes puedan acceder y compartir la información. Se propone que cada protocolo posea las transacciones como funciones que formen parte de sus propios source y header, de esta forma hacer una implementación más general y orientada a los protocolos.

2.2. Propuesta II

2.2.1. Lenguaje de programación

Se propone el desarrollo en el lenguaje de programación Python ya que se puede realizar la implementación de clases para los componentes y posee bibliotecas incluidas para muchas de las herramientas necesarias para la elaboración del proyecto, como la distribución formal, la implementación de un archivo log y el desarrollo de una interfaz de usuario.

2.2.2. Visualización

Se propone el uso de Tkinter, la cual es una biblioteca de desarrollo de interfaz de usuario para Python y está incluida con el lenguaje. Esta herramienta permite una implementación sencilla con todos los componentes necesarios para el desarrollo de las tablas.

2.2.3. Paralelismo

Este lenguaje provee la biblioteca Threading, la cual permite la implementación de hilos para el programa. Se utilizaría un hilo para procesador además de los hilos correspondientes para manejar los protocolos.

2.2.4. Generación de instrucciones

Al igual que en la propuesta anterior se propone una distribución binomial, con esta se puede obtener el tipo de función tomando evaluando cual da como resultado positivo según su probabilidad establecida. Python posee una biblioteca llamada Numpy la cual contiene funciones para cada una de las distribuciones.

2.2.5. Tiempos de ejecución

Se propone manejar la latencia por medio de esperas de tiempo que simule el costo por acceso a memoria. Al igual que en la propuesta anterior se utilizaría como referencia el caso de el sistema Core i7 Xeon mostrado en la sección 1.7, con el acceso a las caché como despreciable y tomando los costos por tiempo y no por ciclos.

2.2.6. Log del sistema

Python cuenta con la biblioteca Logging incluida, esta permite la implementación de un archivo log con distintos tipos de mensajes como de información, advertencias, errores y debug. Esta genera el archivo y muestra los mensajes según su categoría.

2.2.7. Componentes del sistema

Se propone el manejo de componentes por medio de clases en donde cada uno cuente con métodos asociados a las escrituras y los cambios de estado acordes al protocolo siendo implementado. Los procesadores pueden contar con su propio set de instrucciones y funciones de ejecución. La memorias se manejaría como un atributo de la clase y serían manipuladas por medio de sus métodos, que considerarán los

cambios según el protocolo. Se propone una clase que funcione como bus para que las instancias puedan comunicarse entre sí por medio de los atributos de la clase bus.

2.2.8. Protocolos

Cada protocolo contaría con una clase que se encargue del manejo de transacciones entre componentes, por lo que estos controladores funcionan como interfaz entre ellos y los buses deben comunicarse con el controlador para que este tome las acciones necesarias del protocolo.

3. Comparación de propuestas

El proyecto contaba con una especificación muy completa y definida por lo que la mayor diferencia entre las propuestas corresponde a las herramientas utilizadas para la implementación, las cuales fueron mayormente definidas por el lenguaje de programación de cada una, ya que este es que el marca la diferencia más relevante entre ambas.

En la siguiente tabla se hace una comparación de las características de cada propuesta:

	Propuesta I	Propuesta II
Lenguaje de programación	C	Python
Visualización	Manejo de sistema multiprocesador como biblioteca en c utilizada por Python, Tkinter para desarrollo de GUI.	Python Tkinter
Paralelismo	Pthreads	Threading
Generación de instrucciones	Desarrollo de función para distribución binomial.	Distribución binomial de Numpy

Tiempos de ejecución	CLK general y conteo de ciclos de reloj, manejo de latencia por cantidad de ciclos.	Esperas de tiempo por accesos a memoria. Manejo de latencia por tiempo establecido.
Log del sistema	Creación de funciones para manejo del logfile.	Biblioteca Logging.
Componentes del sistema	Structs instanciables para cada memoria y struct de bus de comunicación.	Clases para cada componente y una para la comunicación
Protocolos	Manejo general de cada protocolo por medio de source de cada uno.	Clases de controladores como interfaz de componentes para cada uno de los protocolos.

4. Selección de propuesta

Se toma como selección la Propuesta II por las siguientes razones:

- La propuesta I presenta una mayor dificultad por el hecho de que se debe realizar la implementación de varias funcionalidades para las cuales el lenguaje de la propuesta II ya posee bibliotecas que proveen herramientas con dichas funcionalidades.
- Para la implementación de una interfaz de usuario en la Propuesta I se debe recurrir a otro lenguaje de programación lo que complica su implementación. En cambio la propuesta I posee la herramienta dentro del mismo lenguaje y es ampliamente usada y soportada.
- El manejo de latencia por tiempo me parece más acertado debido a la posibilidad de implementar los retardos de tiempo dentro de las mismas funciones de las clases de la memoria y de la ejecución, en cambio en la propuesta I se hubiera tenido que desarrollar el manejo de los ciclos con el acceso a el reloj del sistema.

- El manejo de los protocolos es mejor hacerlo por medio de una interfaz de componentes desarrollada como clase, de forma que pueda simplemente ser instanciada por cada chip.

5. Implementación de diseño

5.1. Descripción General

Se cuenta con un sistema multiprocesador con una memoria, un controlador para protocolo de directorios y dos chips los cuales contienen una memoria caché L2 un controlador para el protocolo MSI y dos procesadores con una memoria caché L1 cada uno.

En el sistema los procesadores tienen acceso directo a la caché L1, esta hace peticiones de transacciones al controlador del protocolo MSI el cual tiene acceso a la memoria L2 para la ejecución de estas transacciones. A su vez la memoria L2 es el nivel más externo del chip, por lo que de este punto pasa a comunicarse con el controlador de directorios el cual tiene acceso a ambos chip y a la memoria para ejecutar las transacciones del protocolo de directorios y mantener coherencia entre las memorias L2 y la principal. Esta estructura se puede observar en la sección 5.1.

5.2. Diagrama de bloques

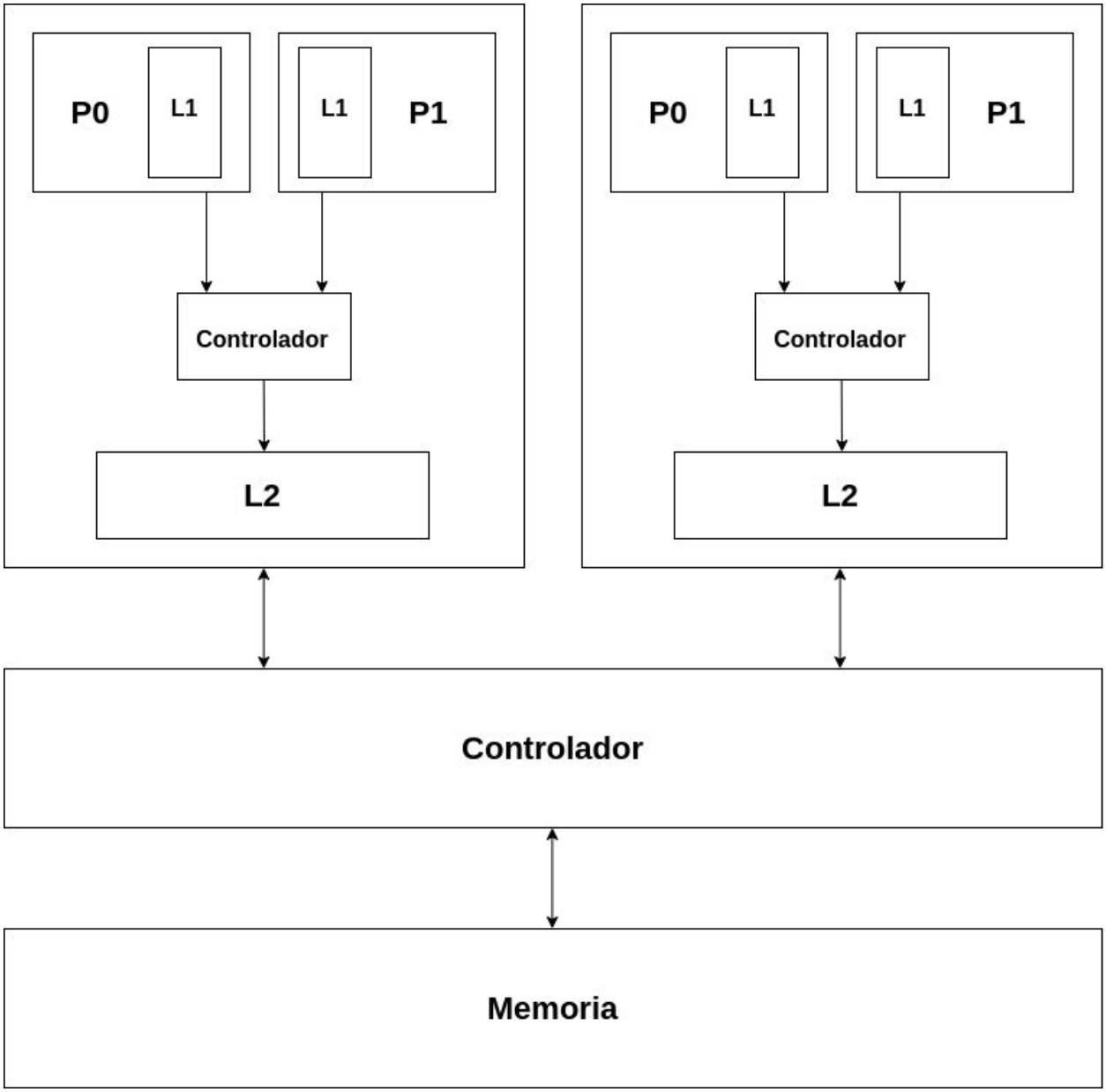

Figura 6. Diagrama de bloques del sistema

5.3. Protocolo MSI

Como ya se explicó en la sección 1.4, el protocolo MSI se maneja por medio de transacciones que manejan las peticiones de escritura y lectura acorde a los estados de cada bloque de caché.

En este caso todas estas transacciones de manejarán por medio de un controlador tal como se muestra en la sección 5.4. Además el estado de los bloques de caché también deben asignarse dentro de las caché según las peticiones que esta reciba.

En la siguiente figura se puede observar el flujo de peticiones desde el procesador hasta la caché L2. Como se muestra en la imagen, el procesador puede hacer dos peticiones a la caché, las cuales corresponden a lectura y escritura. La caché L1 puede realizar 3 peticiones al controlador: BusRd, BusRdX y Bus Upgr; quien a su vez puede realizar peticiones a la memoria caché L2 para obtener el dato en caso de no tenerlo o para escribirlo en esta memoria.

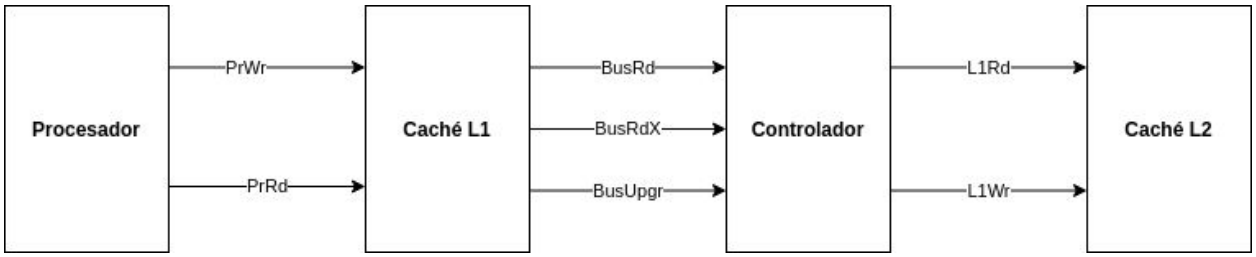

Figura 6. Flujo de transacciones del protocolo MSI

Los estados de los bloques de caché se manejarán de acuerdo al protocolo como se puestra en el diagrama de estados de la sección 1.4. Para tenerlo más claro se pueden observar las clases y sus métodos en la sección 5.5, con la diferencia de que el controlador tiene el nombre de ‘MSI controller’.

5.4. Protocolo basado en directorios

El protocolo basado en directorios implementado es el explicado en la sección 1.5. Este utilizará los mismos estados que el protocolo MSI y será manejado por un controlador entre las caché L2 y la memoria, en donde los directorios estarán dentro de cada memoria en conjunto con la información de cada bloque. El flujo de peticiones se puede observar en la siguiente figura. La caché L2 puede realizar 3 peticiones: BusRd, BusRdX y BusUpgr, el controlador puede realizar dos: L2Rd y L2Wr para lectura y escritura en la memoria y es el encargado de actualizar los estados de cada bloque. Además se añade una petición llamada Mem2L2Upgr implementada para actualizar el estado de 'Compartido Externamente' de los bloques de la caché L2.

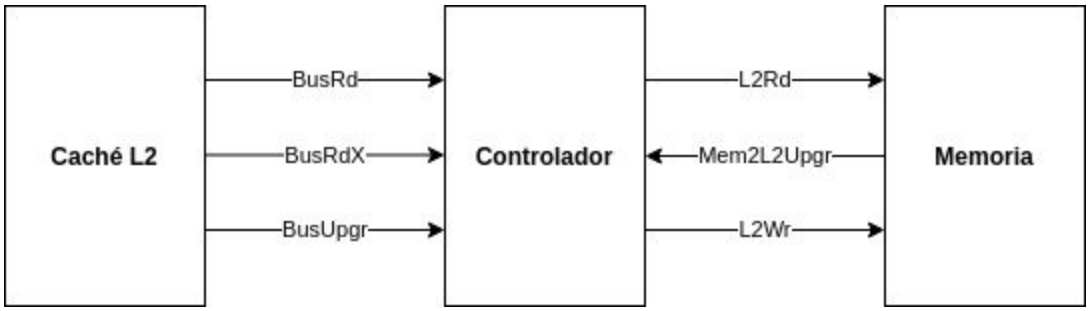

Figura 7. Flujo de transacciones del protocolo basado en directorios

5.5. Diagrama UML

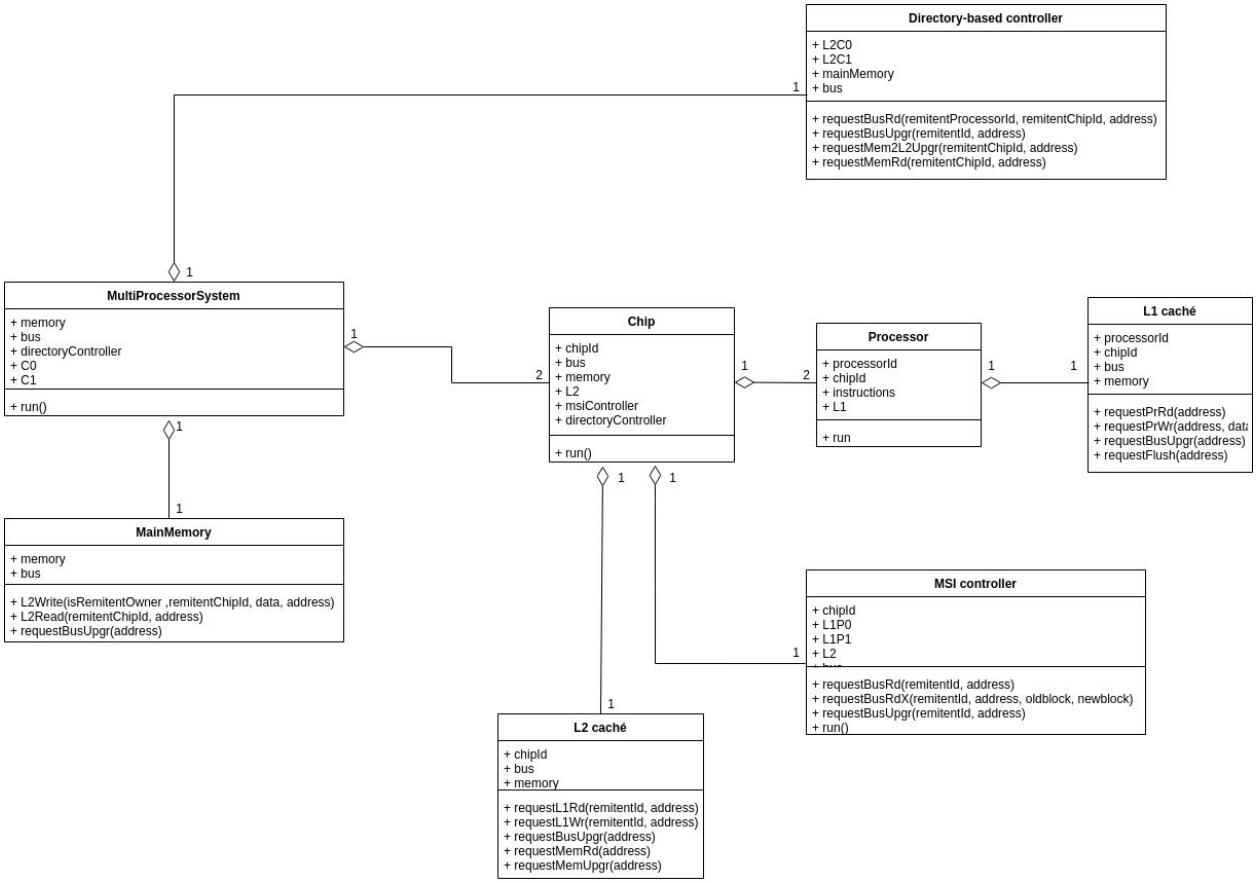

Figura 8. Diagrama UML de la aplicación

5.6. Distribución de probabilidad binomial

Este tipo de distribución provee un resultado binario basado en un valor de probabilidad. Esto permite determinar si una instrucción va a ser de un tipo o no, de manera que en la implementación se evalúa tipo por tipo y cuando la distribución da positivo se agrega este tipo a las instrucciones. Se escogió una probabilidad de 0.4 para instrucciones CALC, esta es la mayor, debido a que por lo general son las más utilizadas y duran menos, sin embargo no se le dio un valor muy alto debido a que el objetivo del proyecto es ver la interacción de las memorias con los protocolos, por lo que tanto a WRITE como a READ se les dio un valor de 0.3.

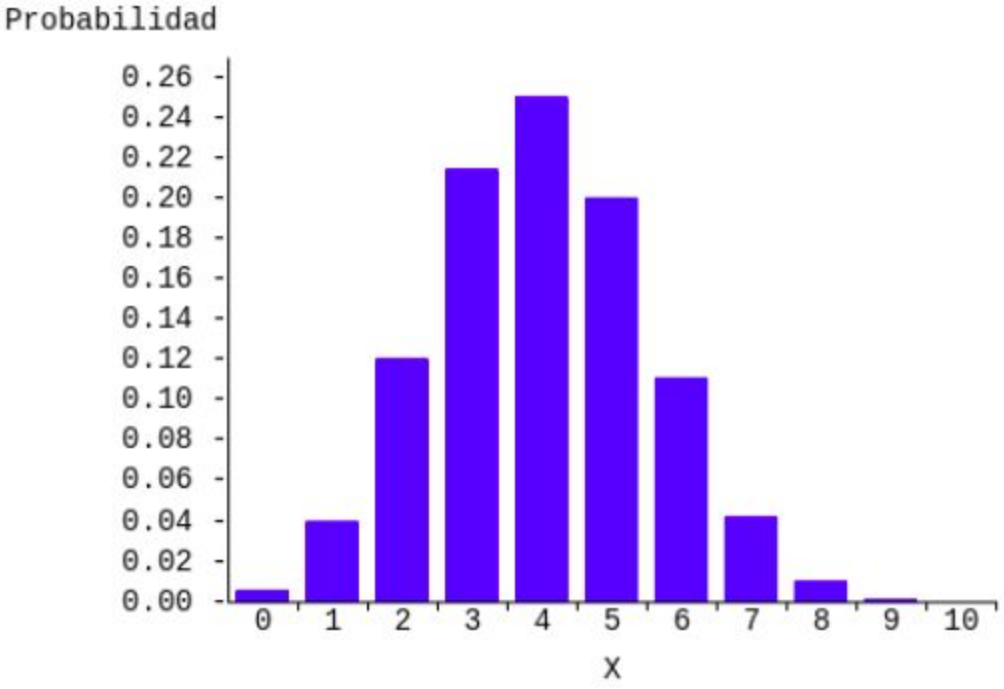

Figura 9. Distribución de probabilidad formal binomial con p=0.4

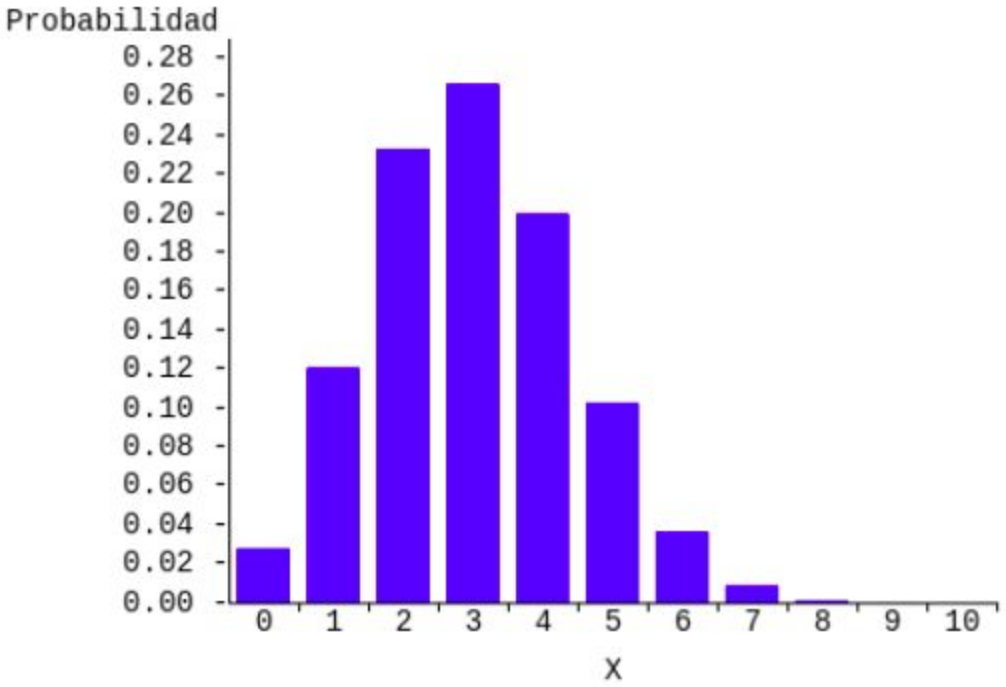

Figura 9. Distribución de probabilidad formal binomial con p=0.3