Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores

Curso: CE4302 – Arquitectura de Computadores II. I Semestre 2022

Profesor: Ing. Luis Barboza Artavia

Estudiante: Jose Alejandro Ibarra Campos - 2018089951

Proyecto Individual – MOESI Simulator: Modelo de protocolo para coherencia de caché en sistemas multiprocesador

Documentación de Diseño

1. Requerimientos del sistema

Número	Requerimiento
1	Debe tener interfaz gráfica para interactuar con este, que muestre en todo momento el estado de todos los componentes del sistema
2	Debe tener cuatro procesadores
3	Cada procesador tiene una memoria caché L1 con cuatro bloques y asociatividad *two-way*. (Inicia en 0)
4	Cada bloque debe tener número, estado de coherencia, dirección y dato
5	El estado de coherencia debe ser asignado según el protocolo de coherencia MOESI
6	Debe existir una memoria principal con 8 bloques (Inicia en 0)
7	Las direcciones de memoria son de 3 bits en binario
8	Los datos son de 4 bits en hexadecimal
9	Los procesadores se deben comunicar con la memoria a través del bus
10	Se deben actualizar las copias según MOESI a través del bus
11	Las instrucciones se generan por procesador, de forma aleatoria, según un generador de distribución binomial o de Poisson
12	Existen tres posibles instrucciones. WRITE dirección;dato, READ dirección, CALC (no hace nada)

13	Se debe simular la penalidad de acceso a memoria en costo de tiempo
14	Los procesadores deben operar en hilos aparte para simular una ejecución simultánea.
15	Debe existir un modo paso a paso, donde se ejecute nada más una instrucción por procesador
16	Se debe permitir agregar instrucciones específicas a cada procesador a través de la interfaz, a ser ejecutadas en el siguiente ciclo.
17	El lenguaje de programación y herramientas de desarrollo son libres
18	El modelado debe ser consistente con los componentes e interacción de estos en el *hardware* real
19	Se debe permitir pausar y reanudar la ejecución de la simulación
20	Se debe permitir la modificación del tiempo base que tarda en ejecutar una instrucción

2. Opciones de solución:

a) Bus único y Python.

Esta propuesta se apoya en la existencia de un bus único que solo puede ser accedido por una instancia a la vez. Esto garantiza una consistencia sencilla en la implementación. Cada CPU en hilo se comunica con este bus si su controlador determina que es necesario obtener algún dato de otra instancia o de memoria. Se contempla también un lenguaje de alto nivel flexible para la interfaz.

En resumen, la idea consiste en los siguientes puntos:

- **Interfaz gráfica**: Se desarrolla la interfaz gráfica en *tkinter* con hilos en *pthread*, debido a su integración nativa. Aquí se interactúa con el simulador y se observa su estado en todo momento.
- **Modelado:** Se modela el sistema con base en la especificación y requerimientos, con un único bus que debe ser bloqueado para que solo una instancia del hilo pueda acceder a él en un mismo instante. Así se garantiza la completitud exitosa de las transacciones sin complicarse.

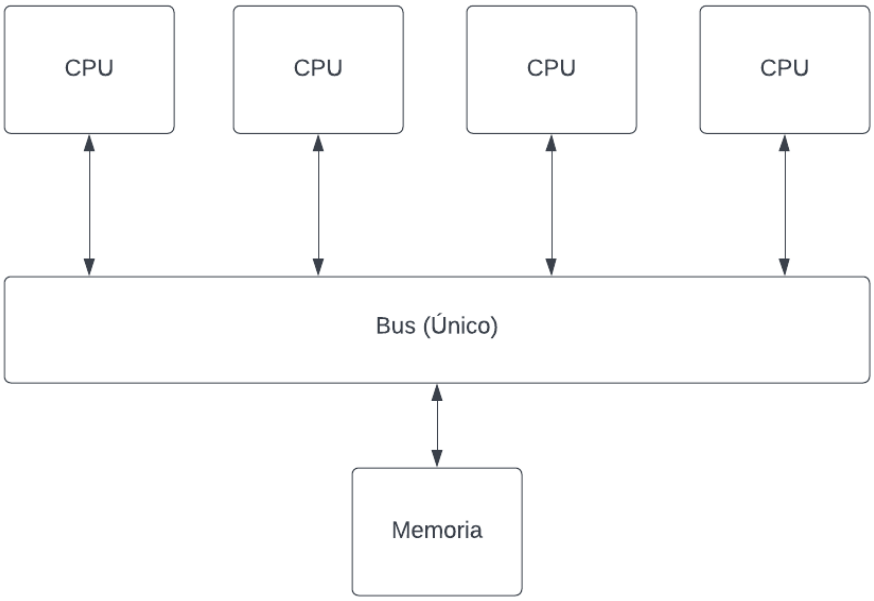

Figura 1. Diagrama de solución a.

b) Bus múltiple y C++

Bajo esta propuesta, se aprovecha el manejo de memoria y eficiencia de C++ para crear un bus múltiple capaz de bloquearse de forma parcial según las conexiones que haga (simulando múltiples cables). Además, se utilizaría la biblioteca poderosa Qt en este lenguaje para la interfaz gráfica.

- **Interfaz gráfica:** Se plantea el uso de Qt, una biblioteca gráfica muy popular, con la capacidad de funcionar de forma dinámica y adaptable para representar el estado de todo el sistema en un instante.
- **Modelado:** Se modela el sistema con un bus múltiple, que se bloquea dependiendo del tipo de conexión u operación a realizar. La implementación es más compleja y utiliza el lenguaje de programación C++.

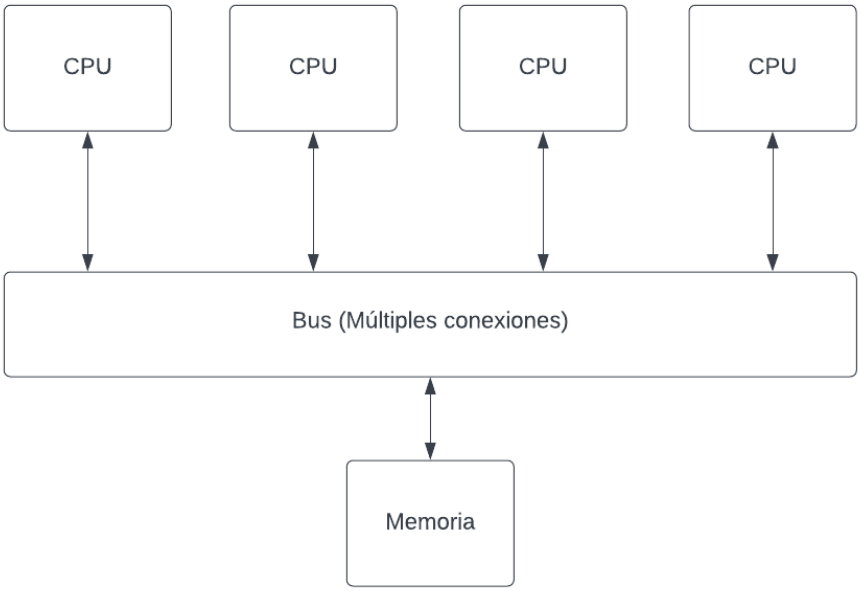

Figura 2. Diagrama de solución b.

3. Comparación de opciones de solución

Entre ambas opciones se destaca la diferencia en el tiempo de desarrollo para cada una, relacionada con la amplia experiencia previa en *Python* en comparación con C++. De primera entrada, parece que se obtendría un mejor resultado con la opción 2, pero se debe considerar que, en términos prácticos, los recursos computacionales no son tan restringidos para la aplicación que se desea. Por esta razón, aunque *Python* sea más lento y poco detallado a nivel de manejo de memoria que C++, se puede obtener una ilusión lo suficientemente convincente del simulador con menos complicaciones en la codificación.

Sucede lo mismo con el bus múltiple. Si bien bajo otro contexto más limitado puede ser la opción más viable, para efectos de este proyecto, cuyo fin es verificar académicamente la comprensión del concepto de MOESI, no es necesario implementar un sistema tan complejo. Con respecto a la interfaz gráfica, Qt es muy poderoso y provee muchísimas herramientas, pero requiere de una configuración previa más costosa. *tkinter*, por el otro

lado, viene integrado de una vez con el ambiente de *Python*. Aunque es un poco tieso, es más sencillo de entender.

4. Selección de la propuesta final

Luego de realizar los análisis anteriormente mencionados, se descartó la opción b, debido a su complejidad y herramientas extra que no serán utilizadas en este proyecto. Además, el diseño de mayor nivel de la opción a permite apreciar mejor lo relevante de las estructuras involucradas en el sistema. De esta manera, se pueden enfocar los esfuerzos en desarrollar de forma correcta y eficiente el control del protocolo de MOESI, que es el enfoque principal de este proyecto.