Requerimientos

1. Necesidades y requerimientos del sistema de simulación de chip multiprocesador y protocolo de coherencia MOESI: Esto incluye definir las funciones y características que el sistema debe tener, como la capacidad de simular la interacción entre cuatro procesadores y la implementación del protocolo de coherencia MOESI.
2. Considerar aspectos de salud y seguridad pública: Es importante considerar aspectos de salud y seguridad pública al diseñar el sistema, especialmente si se espera que el sistema se utilice en aplicaciones críticas. Por ejemplo, el sistema debe ser capaz de manejar errores y fallas de manera segura para evitar cualquier riesgo para los usuarios.
3. Costo total de la vida: El costo total de la vida del sistema debe ser considerado, incluyendo los costos de adquisición, instalación, operación y mantenimiento. Es importante asegurarse de que el sistema sea rentable y sostenible a largo plazo.
4. Carbono neto cero: El impacto ambiental del sistema también debe ser considerado, incluyendo la reducción de emisiones de carbono y la eficiencia energética. Se debe diseñar el sistema de manera que sea lo más eficiente posible en términos de consumo de energía y uso de recursos.
5. Considerar factores culturales, sociales y ambientales según sea necesario: Dependiendo del contexto en el que se utilizará el sistema, puede ser necesario considerar factores culturales, sociales y ambientales. Por ejemplo, si el sistema se utilizará en un entorno multicultural, es importante asegurarse de que la interfaz de usuario sea accesible para todos los usuarios.

Opciones de solución al problema

Se presentan las siguientes opciones a grandes razgos para el modelado del sistema.

1. Diseñar e implementar un modelo de sistema multiprocesador desde 0 utilizando algún lenguaje de programación de alto nivel. Esta es la opción más básica e implica implementar todo desde 0.
2. Utilizar una herramienta de simulación de redes, como NS-3 o Omnet++, para simular el comportamiento de la memoria distribuida y los protocolos de coherencia.
3. Utilizar una herramienta de modelado de sistemas, como MATLAB o Simulink, para diseñar un modelo del sistema multiprocesador y simular su comportamiento.
4. Utilizar una herramienta de simulación de eventos discretos, como AnyLogic o SimPy, para simular el comportamiento de la memoria distribuida y los protocolos de coherencia.

Por los requerimientos del proyecto se implementará el modelo del sistema multiprocesador desde 0 en un lenguaje de alto nivel, ya que las otras opciones podrían infringir sobre lo permitido para los objetivos del curso.

Para la opción de lenguaje se tienen varias opciones:

1. Funcional: Modelar el sistema principalmente a través de funciones en un lenguaje descendiente de Lisp o Haskell.
2. POO: Modelar el sistema utilizando un fuerte enfoque de orientación a objetos en un lenguaje como Java o C++.
3. Imperativo: Modelar el sistema en un lenguaje imperativo, teniendo más libertad sobre el paradigma mismo, utilizando un lenguaje como C, Go o Rust.

También está el tema de comunicación entre hilos, las principales opciones analizadas son:

1. Locks: Comunicarse principalmente a través del bloqueo y desbloqueo de diferentes “locks”, por ejemplo Mutex.
2. Mensajes: Comunicarse enviándose mensajes a través de canales de comunicación, este es el paradigma principal de un lenguaje como Go.
3. Primitivas atómicas: Otro mecanismo sería principalmente a través de primitivas atómicas puras comunicarse creando nuestras propias estructuras de communicación.

Por último para el modelo del sistema hay muchas opciones sobre cómo pueden operar las diferentes unidades. Para el procesador se podrían tener un CPU que ejecuta instrucciones separado del controlador que monitorea el bus de forma independiente, o podría verse como una sola unidad y monitorear el bus es simplemente una de las múltiples tareas del procesador. También el sistema de procesadores + bus + memoria principal. Podría modelarse la memoria principal y el bus como sus propios entes independientes que manejan su propia lógica, o podría modelarse un sistema de control que está encargado de administrar estos recursos.

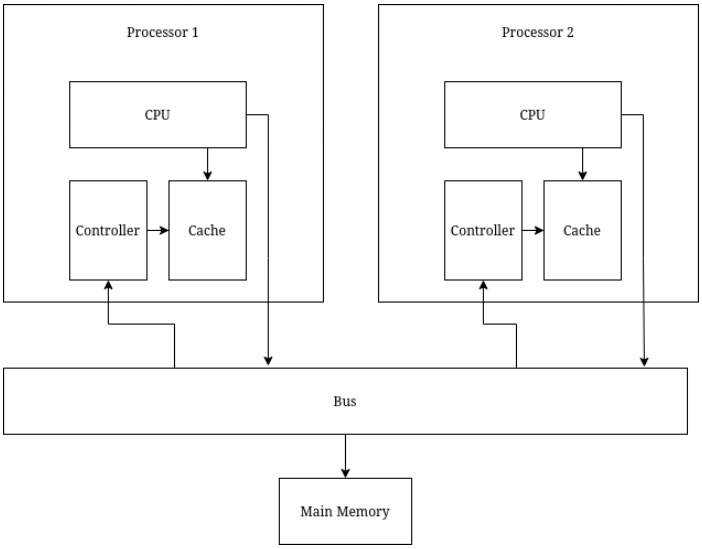

Figura 1: Sistema modelado de forma “transparente” donde cada elemento opera de forma independiente y se conecta de forma directa a los otros elementos

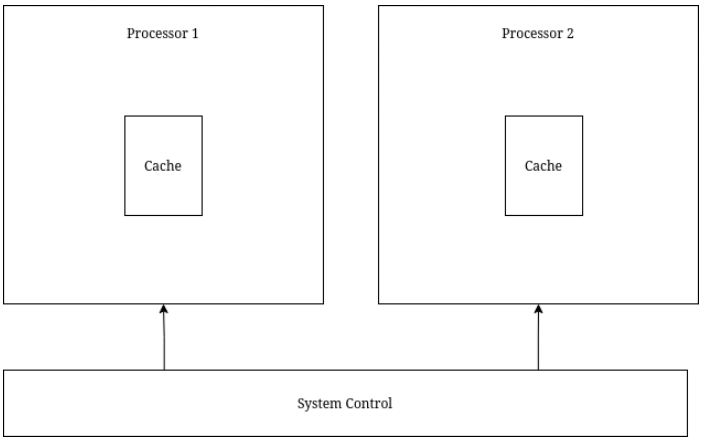

Figura 2: Sistema modelado de forma “opaca” donde los elementos internos a cada subsistema son abstraídos y los subsistemas controlan varias de las tareas

Selección y diseño de propuesta final

Se decide modelar el sistema utilizando el lenguaje Rust ya que este tiene un modelo de hilos muy fuerte con un fuerte enfoque en seguridad de datos y de tipos. Esto puede ahorrar mucho tiempo debuggeando problemas inesperados a futuro en el proyecto y asegura una mejor experiencia más segura para el usuario. También es un lenguaje muy flexible que no está atado a ningún paradigma, permite uso de algunos conceptos de orientación a objetos y otros de paradigma funcional, y al mismo tiempo uno puede construir las cosas desde 0 de una forma básica imperativa como en C.

Otro beneficio de Rust es que es un lenguaje compilado que tiene de los mejores rendimientos medidos, por lo que correr un programa de Rust es de los que menos costo energético tienen ya que demandan menor rendimiento del sistema. Esto especialmente en comparación con algunos lenguajes interpretados como Python o Javascript.

Debido a la escogencia de Rust como lenguaje se decide que para la comunicación entre hilos principalmente se pasarán mensajes. Aunque Rust permite otros métodos para sincronización entre hilos este es el que más promueve.

Para el modelo del sistema se escogió algo mixto, el CPU y el controlador dentro de cada procesador sí son unidades independientes como en el modelo "transparente" y estos tienen conexiones directas al bus. Pero el bus y la memoria principal son controlados por un subsistema de control general. Se muestra detalladamente en la figura 3, tomando en cuenta la generación de instrucciones y el GUI de la aplicación.

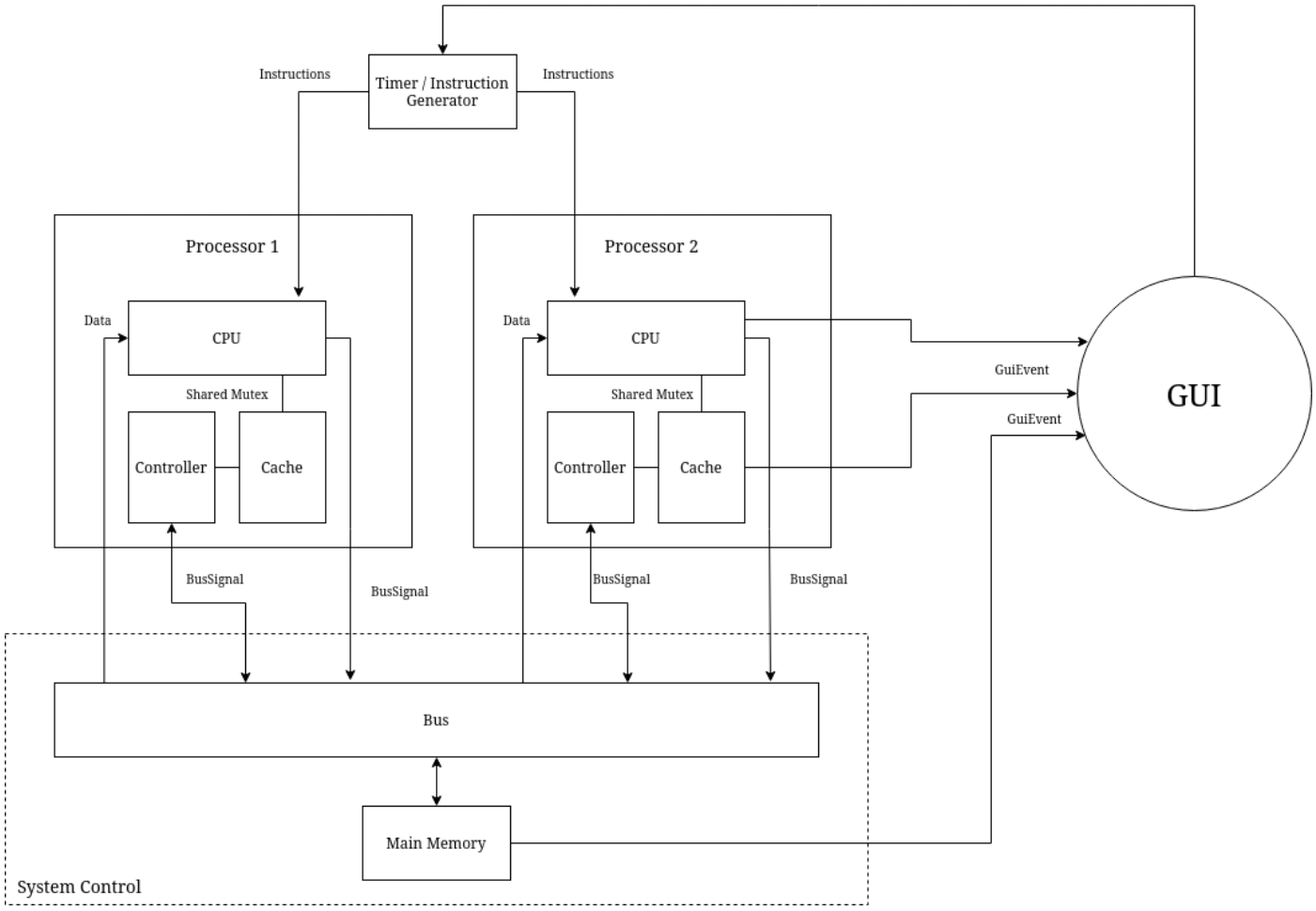

Figura 3: Sistema modelado con controlador y CPU independientes. Sistema de control que maneja los recursos del bus y memoria principal y tomando en cuenta comunicación con la interfaz gráfica. Simplificado para solo mostrar 2 procesadores, pero se puede extender a cualquier cantidad.

Para la implementación del protocolo MOESI se tomaron las siguientes decisiones específicas a esta implementación:

- La política de *write-back* es tal que solo se escribirá un dato de vuelta a la memoria principal si este se encuentra en estado 'O' o 'M' y se necesita guardar otro dato en la caché donde este se encuentra.
- Cuando se hace una escritura en el estado 'O' no se actualizan los valores en estado 'S' en otras cachés, sino que solo se invalidan y el bloque que estaba en estado 'O' pasa a estado 'M'.

- Para la política de reemplazo se le asignó prioridades a los diferentes estados, de mayor a menor prioridad ‘O’, ‘M’, ‘E’, ‘S’, ‘I’. Luego al guardar un nuevo dato en caché (cuya dirección no se encuentre guardada aún, como en un read miss), se remplaza el bloque dentro del set correspondiente a esa dirección que tenga la menor prioridad.

Validación del diseño

Para la validación del diseño se hicieron extensas pruebas ya una vez construida la aplicación, a continuación las pruebas:

- Se dejó correr el sistema por periodos extensos de tiempo en modo automático y se verificó que no parece nunca llegar a fallar.
- Se dejó correr el sistema por periodos extensos de tiempo en modo automático y luego se pausó y verificó que todos los estados mostrados fueran válidos y correctos.
- Se corrió el sistema en modo manual y se verificó que todas las transiciones de estados y operaciones fueran las correctas.
- Se realizaron diversas pruebas manuales dándole al sistema instrucciones escogidas de forma específica para verificar todas las transiciones de estados conocidas.