TEC | Tecnológico de Costa Rica

Modelo de protocolo para coherencia de caché en sistemas multiprocesador

Área Académica Ingeniería en Computadores
CE4302 — Arquitectura de Computadores II

Estudiante:
Carlos Adrián Araya Ramírez

Profesor. Luis Barboza Artavia

Cartago, Costa Rica

Abril, 2023

Índice

1. Descripción **2**

2. Listado de requerimientos del sistema **3**

3. Elaboración de opciones de solución al problema **4**
3.1. Propuesta 1 4
3.2. Propuesta 2: 6

4. Valoración de opciones de solución **10**
4.1. Propuesta 1 10
4.2. Propuesta 2 10

5. Selección de la propuesta final **10**

6. Diseño de la alternativa seleccionada **11**
6.1. Clases utilizadas en el modelado y la abstracción del sistema: 13
6.1.1. CPU Core 13
6.1.2. CPU Controller: 14
6.1.3. Bus 15
6.1.4. Main Memory 16
6.1.5. Utils 17
6.2. Justificación de la elección de la distribución de probabilidad de Poisson para la generación de instrucciones aleatorias 18

7. Validación del diseño **18**

1. Descripción

Este documento presenta una descripción detallada de la implementación de un simulador del protocolo de coherencia de caché *MOESI* para sistemas multiprocesador. El modelo se compone de un sistema multiprocesador de cuatro núcleos, una memoria principal que cuenta con ocho bloques, y un bus que permite la comunicación entre la memoria y cada uno de los núcleos.

Cada núcleo del procesador cuenta con un controlador y una caché L1 de cuatro bloques. La implementación del simulador debe emplear la asociatividad por set two-way y debe tener la capacidad de procesar las instrucciones *CALC*, *READ* y *WRITE*.

2. Listado de requerimientos del sistema

En esta sección se detallan los requerimientos planteados para solucionar el problema tomando en consideración factores como la seguridad, el costo total de la vida y el carbono neto cero.

1. Se requiere tomar como objetivo principal optimización de la implementación del simulador para que consuma la menor cantidad de recursos computacionales posible, en línea con el objetivo de reducir la huella de carbono.

2. Se requiere implementar en la interfaz del programa cuatro tablas donde cada una represente la caché L1 de cada núcleo del procesador asegurándose de que la interfaz sea clara y fácil de entender para el usuario y que muestre información relevante para el monitoreo y la optimización del rendimiento del procesador.

3. Se requiere implementar en la interfaz del programa una tabla que represente la memoria principal.

4. Se requiere implementar en la interfaz del programa una etiqueta para cada núcleo que muestre los aciertos y desaciertos en caché cada vez que se procesa una instrucción.

5. Se requiere crear un módulo para los núcleos del procesador que tenga un identificador único, una estructura que represente una caché de 4 bloques y métodos que permitan modificar los bloques de la caché y representar de manera gráfica los cambios en tiempo real. Además, se debe implementar mecanismos de seguridad para garantizar la integridad de los datos y evitar fallos en el sistema.

6. Se requiere crear un módulo para la memoria principal que tenga una estructura que represente la memoria principal unificada de 8 bloques y métodos que permitan modificar los bloques de la caché y representar de manera gráfica los cambios en tiempo real.

7. Se requiere crear un módulo controlador para cada procesador que sea capaz de determinar si un dato requerido se encuentra en la caché local y en caso de encontrarlo generar una solicitud para el bus. Se deben incluir medidas de seguridad para evitar colisiones en el bus y garantizar la coherencia de los datos.

8. Se requiere crear un módulo para el bus que sea capaz de procesar los solicitudes generadas por los núcleos y brindarles el dato realizando las operaciones que se requieran según la situación en base al protocolo de coherencia *MOESI* (*Write back*, cambios de estado, etc).

9. Se requiere crear un módulo capaz de generar instrucciones aleatorias para cada núcleo del procesador utilizando una distribución de probabilidad. Este módulo debe estar diseñado para asegurar la aleatoriedad y evitar patrones predecibles que puedan afectar el rendimiento del procesador.

10. Se requiere que el simulador tenga el modo de ejecución paso a paso donde se procesa únicamente una instrucción por cada núcleo del procesador.

11. Se requiere que el simulador tenga el modo de ejecución automático, donde se procesa una instrucción por núcleo secuencialmente hasta que se le indique que se detenga.

3. Elaboración de opciones de solución al problema

3.1. Propuesta 1

En esta primera propuesta la idea es utilizar un hilos y colas para la generación y el procesamiento de instrucciones. Para esta propuesta se utilizará el lenguaje de programación Python en la versión 3.10. A continuación se detalla el diseño de esta propuesta:

1. Crear una clase Bus que tenga las siguientes propiedades:

- Una cola requests que contendrá las solicitudes generadas por los procesadores.
- Una lista caches que contendrá las cachés de cada procesador.
- Un objeto main_memory que represente la memoria principal.

2. Crear un método en la clase Bus que se encargará de procesar las solicitudes que lleguen a la cola requests. Este método deberá:

- Tomar un elemento de la cola de solicitudes.
- Analizar la solicitud para determinar el identificador del núcleo que emitió la solicitud, el tipo de operación (CALC, lectura o escritura) y la dirección de memoria a la que se refiere.
- Realizar la operación en la caché correspondiente y actualizar el estado del bloque de caché según el protocolo MOESI.
- Si es una operación de escritura, propagar los cambios a las demás cachés que tengan copias del bloque en cuestión.

3. Crear una clase Processor que represente a cada uno de los procesadores y utilizar un hilo para cada instancia. Esta clase deberá tener las siguientes propiedades:

- Una propiedad cache que represente su caché.
- Un controlador que determine si requiere generar una solicitud para el bus.

- Métodos para actualizar la caché y mostrar visualmente el cambio utilizando colores.

4. Crear un método en la clase Processor que genere instrucciones aleatorias y las procese localmente si es posible (read hit, write hit) o que las agregue a la cola de solicitudes del bus en los casos que se requiera (read miss, write miss). Este método deberá:

- Generar una dirección de memoria aleatoria.
- Generar un tipo de operación aleatorio (CALC, lectura o escritura).
- Crear una solicitud y agregarla a la cola de solicitudes del bus cuando asi se requiera.

5. Crear una instancia de la clase Bus y cuatro instancias de la clase Processor que se conecten al bus.

Para implementar esta propuesta, se pueden utilizar hilos y colas de la siguiente manera:

- Crear una cola request_queue en el Bus que contendrá todas las solicitudes generadas por los procesadores.
- Crear un hilo que ejecute el método para procesar solicitudes de la cola request_queue de la clase Bus.
- Crear un hilo para cada instancia de la clase Processor. Cada uno de estos hilos ejecutará el método que generará solicitudes aleatorias y las agregará a la cola request_queue.

A continuación, en la figura 1 se muestra un diagrama del diseño de esta propuesta, donde se pueden observar los diferentes módulos y pasos a seguir que se deben implementar en caso de seleccionar esta propuesta.

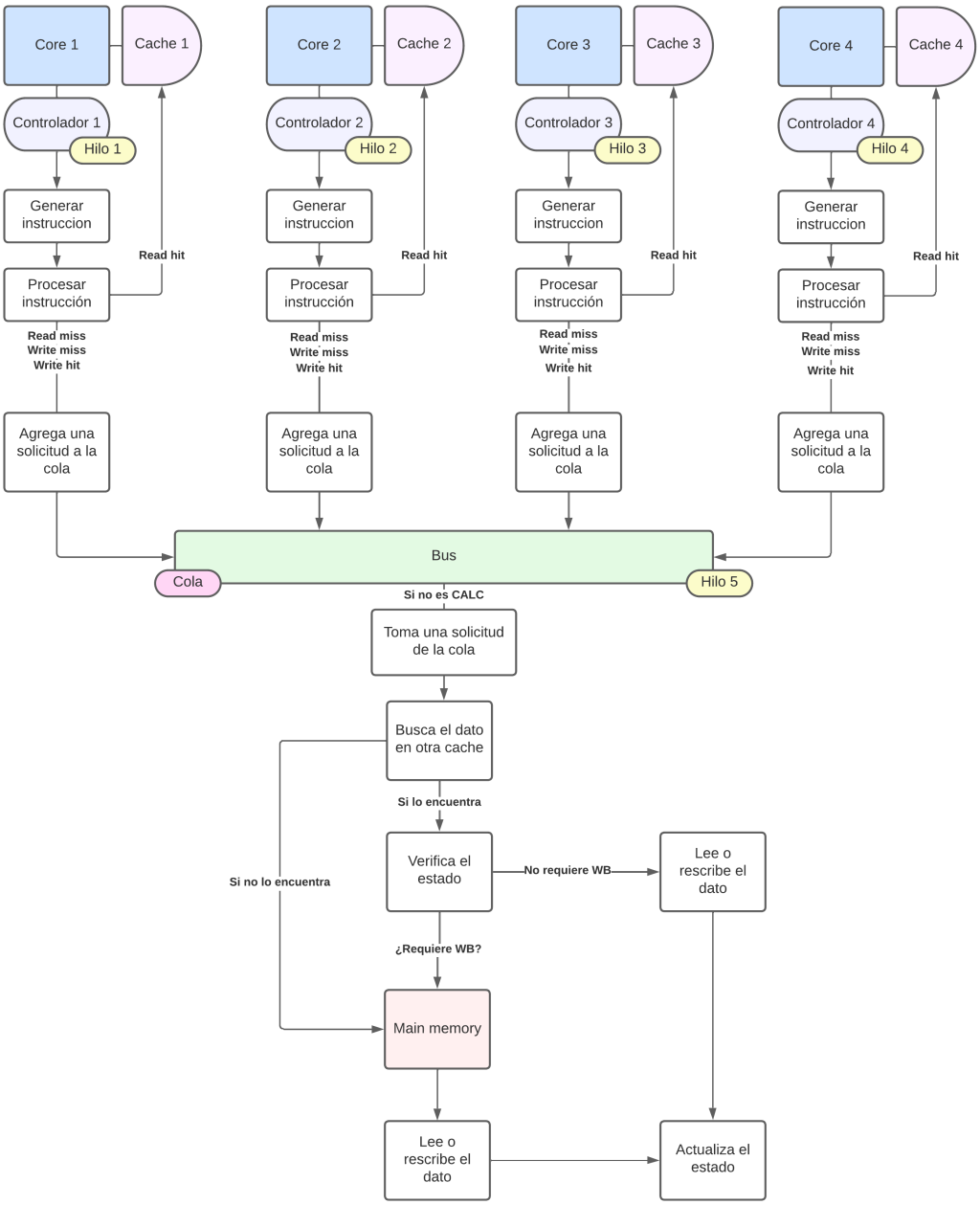

Figura 1: Diagrama de la propuesta 1.

3.2. Propuesta 2:

En esta propuesta alternativa la idea es utilizar un enfoque basado en eventos. Este enfoque implica crear eventos para cada solicitud de CALC, lectura o escritura y programarlos en una cola

de eventos ordenados por su tiempo de ocurrencia. Cada evento tendrá la información necesaria para que el controlador del bus procese la solicitud correspondiente. Para esta propuesta también se utilizará el lenguaje de programación Python en la versión 3.10. A continuación se detalla el diseño de esta propuesta:

1. Crear una clase Evento que tenga las siguientes propiedades:

- Un tiempo de ocurrencia (timestamp) que indique el momento en que se debe procesar el evento.
- Una dirección de memoria que identifique la ubicación de la información que se quiere leer o escribir.
- Un tipo de operación (CALC, lectura o escritura) que indique la operación que se desea realizar.
- Un identificador del procesador que originó la solicitud.

2. Crear una clase Bus que tenga las siguientes propiedades:

- Una lista de cachés (caches) que contienen los bloques de memoria en caché de cada procesador.
- Un objeto main_memory que represente la memoria principal.
- Un objeto lock para garantizar la exclusión mutua en el acceso al bus.
- Un tiempo de sistema (system_time) que indique el momento actual del sistema.

3. Crear un método schedule_event en la clase Bus que agregue un evento a la cola de eventos, ordenándolos por su tiempo de ocurrencia.

4. Crear un método run en la clase Bus que se encargue de procesar los eventos de la cola de eventos. Este método deberá:

- Tomar el primer evento de la cola de eventos y bloquear el bus con el objeto lock.
- Analizar el evento para determinar el tipo de operación y la dirección de memoria a la que se refiere.
- Realizar la operación en la caché correspondiente y actualizar el estado del bloque de caché según el protocolo MOESI.
- Si es una operación de escritura, propagar los cambios a las demás cachés que tengan copias del bloque en cuestión.

- Desbloquear el bus con el objeto lock.
- Incrementar el tiempo de sistema (system_time).

5. Crear una clase Processor que represente a cada uno de los procesadores. Esta clase deberá tener las siguientes propiedades:

- Una propiedad cache que represente su caché.
- Un objeto bus que represente el bus al que se conectará el procesador.
- Un identificador único (processor_id) que identifique al procesador.

6. Crear un método generate_request en la clase Processor que genere una solicitud de lectura o escritura y la agregue a la cola de eventos del bus. Este método deberá:

- Generar una dirección de memoria aleatoria.
- Generar un tipo de operación aleatorio (CALC, lectura o escritura).
- Crear un evento con la información de la solicitud y agregarlo a la cola de eventos del bus utilizando el método schedule_event.

7. Crear una instancia de la clase Bus y cuatro instancias de la clase Processor que se conecten al bus.

A continuación, en la figura 2 se muestra un diagrama del diseño de esta propuesta, donde se pueden observar los diferentes módulos y pasos a seguir que se deben implementar en caso de seleccionar esta propuesta.

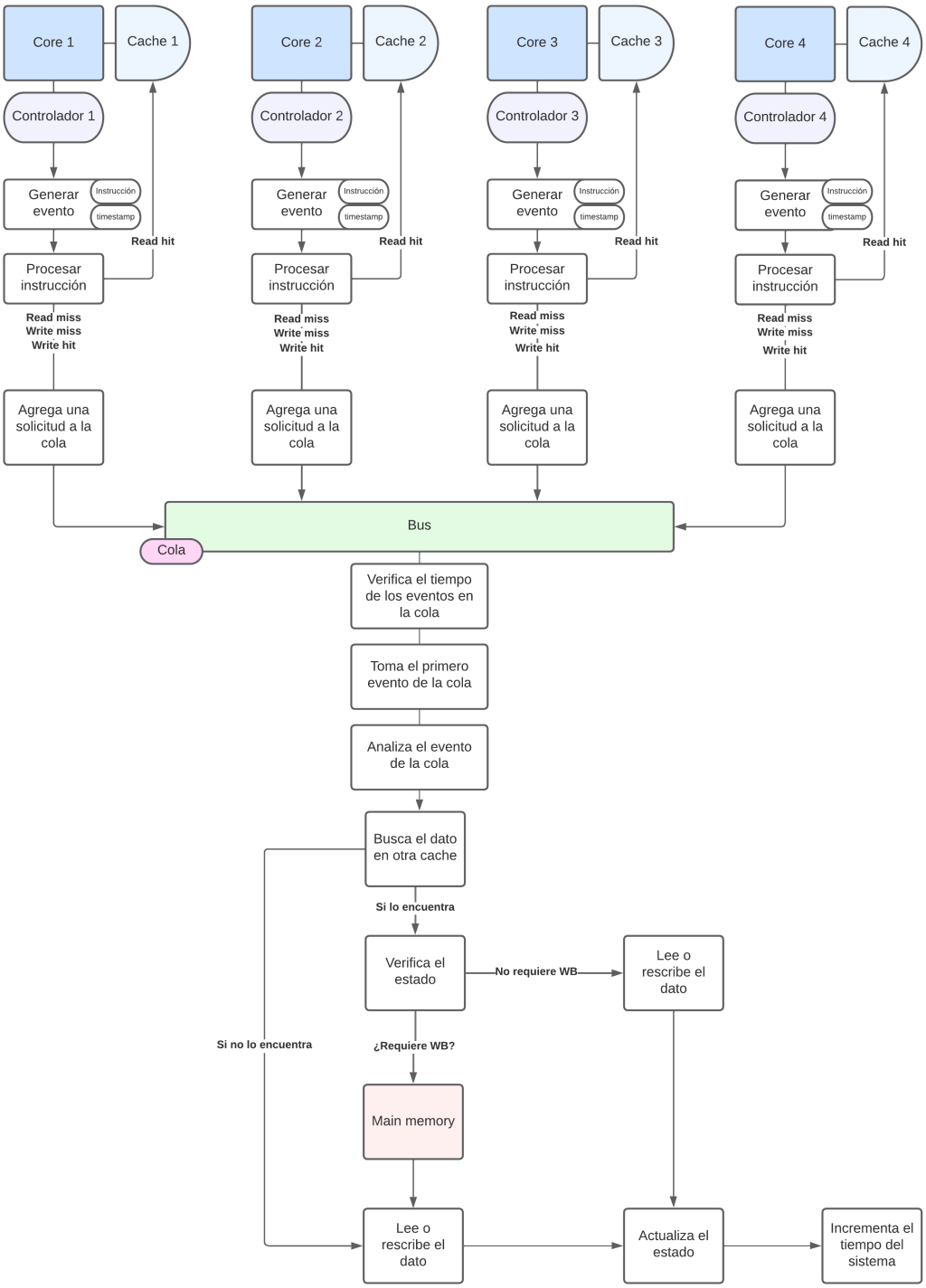

Figura 2: Diagrama de la propuesta 2.

4. Valoración de opciones de solución

4.1. Propuesta 1

Esta propuesta utiliza hilos y colas para simular los procesos en paralelo, lo que puede ser más difícil de implementar correctamente y depurar en comparación con la segunda propuesta. Además, el uso de hilos puede generar problemas de sincronización y acceso a datos compartidos, lo que podría afectar el rendimiento y la precisión de la simulación.

Aunque es cierto que esta propuesta implica un mayor costo computacional que se refleja directamente en aspectos como el costo total de vida, la neutralidad de carbono y el impacto ambiental, es más fácil de implementar. Dado que el problema a resolver es muy complejo, es crucial considerar si vale la pena invertir en una implementación más sólida, eficiente y optimizada. Todo esto, teniendo en cuenta que hay un plazo límite para el desarrollo.

4.2. Propuesta 2

Por otro lado, en esta alternativa no se utilizan hilos ni colas globales, en su lugar se usa una cola de eventos para coordinar el procesamiento de solicitudes del bus. De esta manera, se puede evitar problemas de concurrencia y mejorar la eficiencia de la simulación. Si lo que se desea realmente es crear una solución eficiente y optimizada, definitivamente se debe seleccionar esta propuesta.

Sin embargo, esta solución es más compleja que la solución de la propuesta 1 ya que requiere agregar un módulo extra que se encargue de la generación eventos y el manejo de un nuevo modelo de tiempo que le permita al controlador del bus saber cual solicitud debe procesar en cada momento y como se mencionó anteriormente, la complejidad de la solución juega un papel importante debido a la limitación del tiempo de desarrollo.

5. Selección de la propuesta final

La propuesta 1 fue seleccionada debido a que, aunque implica un mayor costo computacional y presenta problemas potenciales de sincronización y acceso a datos compartidos, es más fácil de implementar y depurar en comparación con la propuesta 2.

Teniendo en cuenta que el problema a resolver es muy complejo y hay un plazo límite para el desarrollo, es importante elegir una solución que sea sólida y eficiente, pero que también se pueda implementar dentro del tiempo disponible. Aunque la propuesta 2 ofrece una solución más eficiente y optimizada, su complejidad adicional podría hacer que su implementación tome más tiempo del disponible.

6. Diseño de la alternativa seleccionada

A continuación se muestra el diseño final de la alternativa seleccionada, primeramente en la figura 3 se muestra el diagrama de bloques de la alternativa seleccionada en donde se observa el modelo completo con los 4 procesadores con su respectiva caché, el bus y la memoria principal.

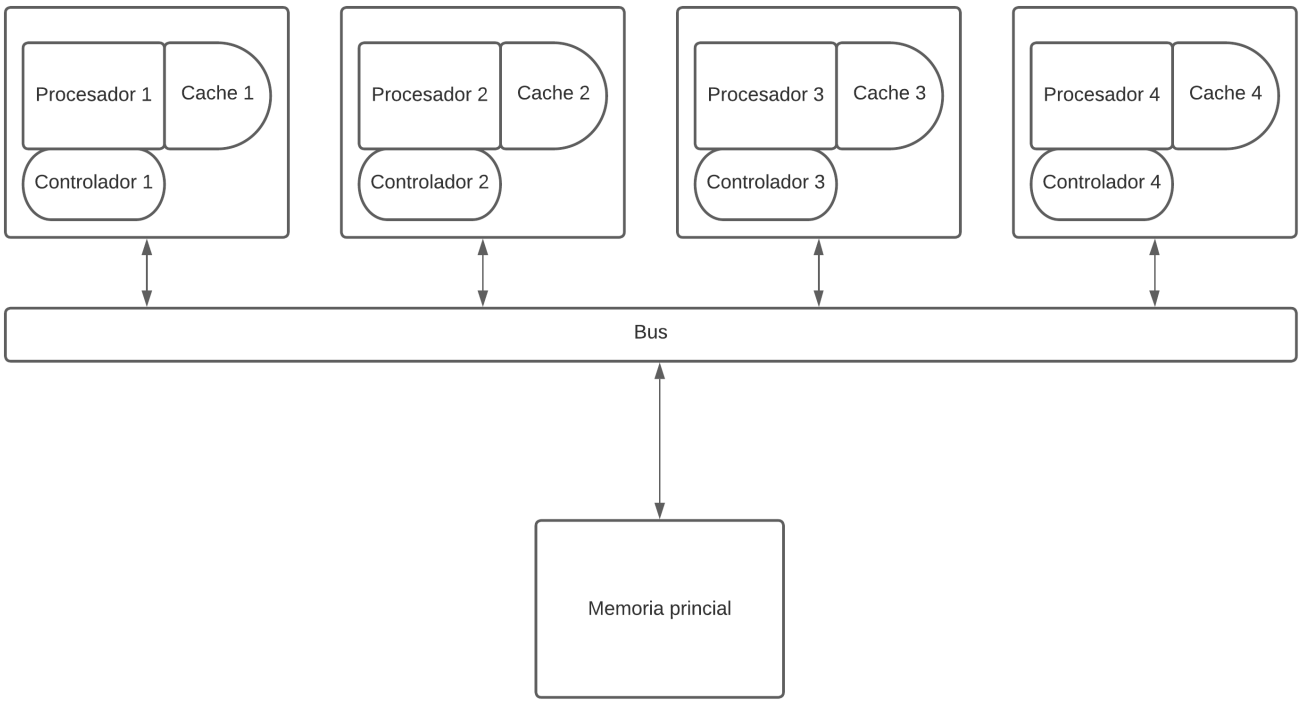

Figura 3: Diagrama de bloques de la alternativa seleccionada.

Ahora en la figura 4 se muestra un mockup de la interfaz que se implementará para el diseño de la alternativa seleccionada, en este mockup se puede observar detalladamente la estructura que tendrá el sistema y los botones para el control del flujo del programa.

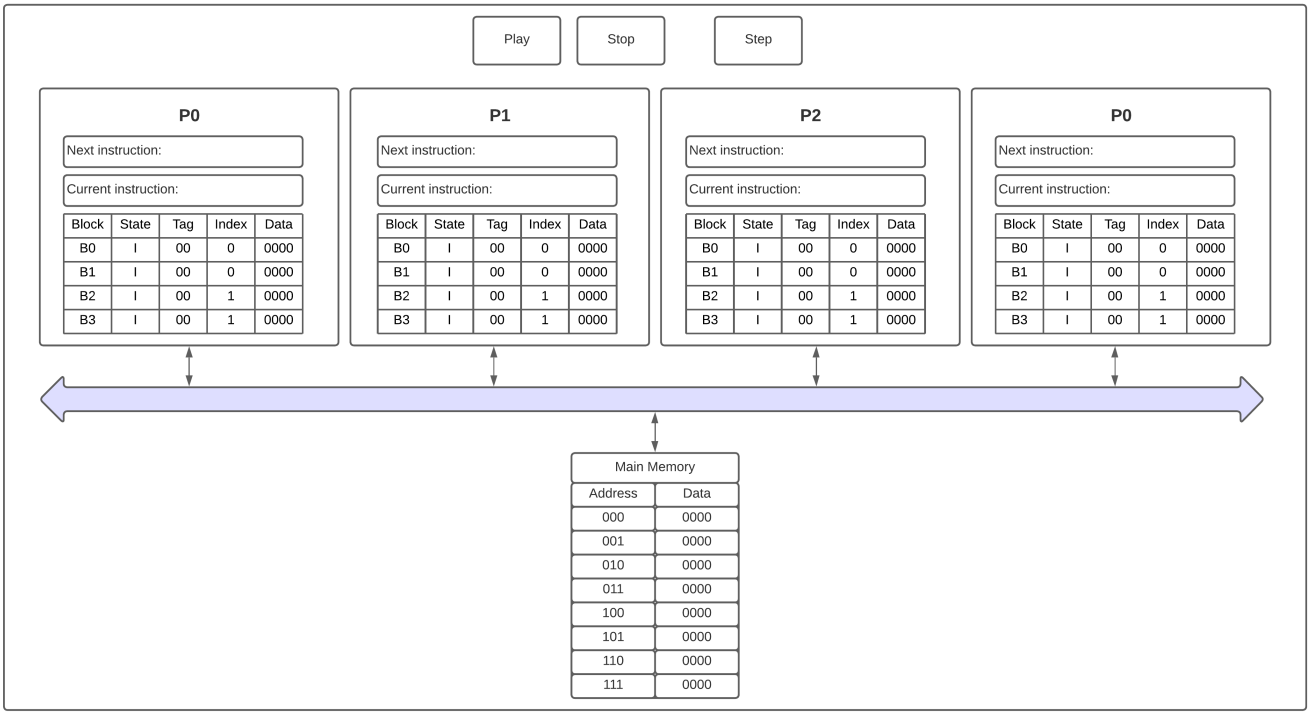

Figura 4: Mockup de la interfaz de la alternativa seleccionada.

Ahora detallarán las clases que se utilizaron para modelar el sistema. En la figura 5 se muestra el diagrama de clases correspondiente a las clases utilizadas en el diseño final.

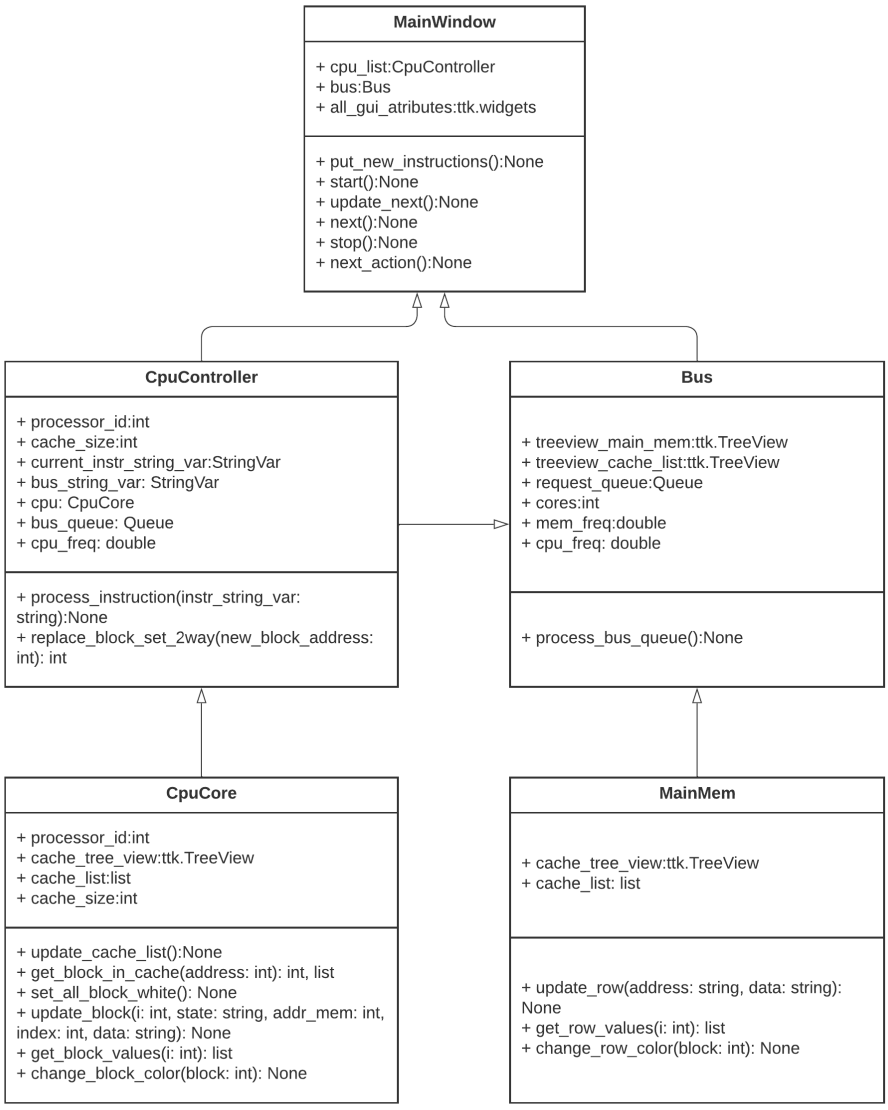

Figura 5: Diagrama de clases de la alternativa seleccionada.

6.1. Clases utilizadas en el modelado y la abstracción del sistema:

6.1.1. CPU Core

Esta clase tiene los siguientes atributos:

- processor_id es un identificador entero el cual se utiliza para diferenciar las cuatro instancias de procesador que tiene el sistema.

- cache_tree_view es un objeto tipo ttk.Treeview de la libreria Tkinter y es utilizado para representar gráficamente una tabla que posee los cuatro bloques de caché del procesador.
- cache_list es una lista que almacena los valores que muestra el objeto ttk.TreeView, a fin de cuentas, el ttk.TreeView es la caché representada en la interfaz y la lista cache_list es utilizada para procesar los datos en la lógica.
- cache_size representa la cantidad de bloques que tiene la caché y es utilizado cuando se requiere iterar sobre los bloques de la caché.

Esta clase tiene los siguientes métodos:

- update_cache_list: mapea los valores que tiene la lista cache_list al ttk.Treeview con el objetivo de mantener actualizada la lógica con la interfaz.
- get_block_in_cache: busca el dato de la dirección pasada por parámetro en su caché y lo retorna si lo cuenta
- update_block: actualiza un bloque de la caché con los nuevos valores pasados por parámetro y actualiza el ttk.TreeView
- get_block_values: retorna los valores de un bloque de caché
- Finalmente se tiene múltiples métodos que cambian el color de un bloque de caché con el objetivo de simular una animación cuando se procesan las instrucciones en ejecución.

6.1.2. CPU Controller:

Esta clase tiene los siguientes atributos:

- processor_id: el identificador de la CPU.
- cache_size: el tamaño de la caché, que se inicializa en 4.
- current_instr_string_var: una variable StringVar que representa la instrucción actual que se está procesando.
- bus_string_var: una variable de StringVar que representa el estado actual del bus.
- cpu: una instancia de la clase CpuCore, que representa el núcleo de la CPU.
- bus_queue: una cola que representa el bus del sistema de memoria caché.
- cpu_freq: la frecuencia de la CPU.

Esta clase tiene los siguientes métodos:

- process_instruction, toma como entrada una cadena que representa una instrucción y realiza las siguientes acciones:
  - Actualiza el valor de current_instr_string_var con la cadena de la instrucción actual.
  - Actualiza la lista de bloques de caché en la CPU y los colorea de blanco.
  - Divide la instrucción en partes.
  - Si la instrucción es una instrucción CALC, actualiza bus_string_var y agrega un mensaje a bus_queue que indica que la CPU está realizando un cálculo.
  - Si la instrucción es una instrucción READ o WRITE, busca el bloque de caché correspondiente a la dirección de memoria en la instrucción en la lista de bloques de caché de la CPU. Si se encuentra el bloque, se determina si es un READ HIT o un WRITE HIT. Si no se encuentra el bloque, se reemplaza el bloque según la política de reemplazo definida en replace_block_set_2way y se indica que es un READ MISS o un WRITE MISS. A continuación, se actualiza bus_string_var con el tipo de solicitud y se agrega un mensaje a bus_queue que indica la CPU que realiza la solicitud, el tipo de solicitud, el número de bloque y la dirección de memoria. Finalmente, el método espera cpu_freq segundos antes de terminar.
- replace_block_set_2way: toma como entrada la dirección de memoria de un bloque que necesita ser reemplazado en la caché y devuelve el índice del bloque que se debe reemplazar según la política de reemplazo definida. La política de reemplazo se basa en los estados I, S, E, M y O, donde I es el estado de bloque más comúnmente reemplazado y O es el estado de bloque menos reemplazado. El método busca el primer bloque en la caché que tenga un estado de I, y si no lo encuentra, busca un bloque en el siguiente estado de la política. Si no encuentra un bloque en ninguno de los estados de la política, devuelve el índice del bloque más antiguo en la caché.

6.1.3. Bus

Esta clase tiene los siguientes atributos:

- treeview_main_mem: objeto de la clase TreeView de la biblioteca Tkinter que representa la memoria principal del simulador de cache.
- bus_queue: cola de solicitudes de acceso al bus de sistema.
- treeview_cache_list: lista de objetos de la clase TreeView que representan las caches de los procesadores. cores: número de procesadores en el sistema.

- mem_freq: frecuencia de acceso a la memoria principal.
- cpu_freq: frecuencia de acceso a la cache del procesador.

Esta clase tiene los siguientes métodos:

- process_bus_queue: procesa las solicitudes de acceso al bus de sistema que hay en la cola. Este método toma el primer elemento de la cola y lo procesa hasta que la cola se vacía. En cada iteración, verifica si la operación solicitada es una operación de lectura o escritura, y en base a eso, realiza diferentes acciones. En el caso de una operación de escritura, también verifica si el bloque de cache en cuestión se encuentra en un estado determinado y, de ser así, realiza las acciones necesarias para llevarlo al estado correcto.
- make_write_back: método que simula una operación de write back en el sistema. Recibe como parámetro un bloque de cache y lo actualiza en la memoria principal.
- search_cache_modified_owned: método que busca en las caches de los procesadores si algún otro procesador tiene el bloque de cache solicitado en un estado de modificación u propiedad. Si lo encuentra, actualiza el estado del bloque en cuestión y lo devuelve. De lo contrario, devuelve None.
- address_is_exclusive: método que verifica si la dirección del bloque de cache solicitado es elegible para ser un bloque en estado exclusivo. Devuelve True si la dirección es elegible y False en caso contrario.
- search_cache_to_invalidate: método que busca en las caches de los procesadores si hay algún otro bloque con la misma dirección que el bloque solicitado y lo invalida. Es decir, lo lleva a un estado de inválido.

6.1.4. Main Memory

Esta clase tiene los siguientes atributos:

- cache_tree_view: es un objeto que se utiliza para representar la vista en árbol de la caché.

Esta clase tiene los siguientes métodos:

- update_row: es un método que se utiliza para actualizar una fila en la vista en árbol de la caché. Recibe dos parámetros, *address* y *data*, que se utilizan para actualizar los valores de la fila correspondiente en la vista en árbol.
- get_row_values: es un método que se utiliza para obtener los valores de una fila en la vista en árbol de la caché. Recibe un parámetro i que representa el índice de la fila que se quiere obtener

y devuelve los valores de la fila correspondiente en forma de una lista.

- change_row_color_green: es un método que se utiliza para cambiar el color de fondo de una fila en la vista en árbol de la caché a verde. Recibe un parámetro *block* que representa el bloque de memoria que se quiere resaltar y utiliza un identificador de etiqueta *m* seguido del número del bloque para identificar la fila correspondiente.

- change_row_color_white: es un método que se utiliza para cambiar el color de fondo de una fila en la vista en árbol de la caché a blanco. Recibe un parámetro *block* que representa el bloque de memoria que se quiere desmarcar y utiliza un identificador de etiqueta *m* seguido del número del bloque para identificar la fila correspondiente.

6.1.5. Utils

Esta clase tiene los siguientes métodos:

- poisson: este método genera un arreglo de números aleatorios de tamaño *size* a partir de una distribución de Poisson con tasa *lam*. En cada iteración del bucle for, el método calcula un valor k utilizando el algoritmo de la distribución de Poisson y lo agrega al arreglo. El método devuelve el arreglo resultante.

- generate_instruction: este método genera una instrucción aleatoria para un procesador identificado por processor_id. Primero se define una lista de operaciones posibles (instr_op) y se utiliza el método *poisson* para generar una lista de números aleatorios que se utilizan para seleccionar una de las operaciones. Si la operación es *READ* o *WRITE*, se genera una dirección aleatoria utilizando el método *poisson*. Si la operación es *WRITE*, también se genera un dato aleatorio en formato hexadecimal. Finalmente, el método devuelve una cadena de texto que representa la instrucción generada.

- set_next_instruction: este método se utiliza para establecer la siguiente instrucción para un procesador identificado por processor_id. El método utiliza el método generate_instruction para generar una instrucción aleatoria y la establece como el valor de la variable next_inst_string_var.

- int_to_binary: este método convierte un número entero *n* en una cadena de texto que representa su valor en binario utilizando 3 dígitos. El método utiliza un bucle while para calcular el valor binario de *n* y lo almacena en la cadena binary_str. Luego, el método agrega ceros a la izquierda a la cadena para que tenga una longitud de 3 dígitos y la devuelve.

6.2. Justificación de la elección de la distribución de probabilidad de Poisson para la generación de instrucciones aleatorias

La distribución de Poisson se utiliza comúnmente para modelar eventos raros que ocurren en un período de tiempo fijo. En este caso, se utiliza para generar instrucciones aleatorias para un procesador. La tasa lambda se establece en 10, lo que indica que se espera que ocurran 10 eventos por unidad de tiempo.

La implementación de la distribución de Poisson en este caso utiliza un algoritmo de aceptación-rechazo para generar números aleatorios a partir de la distribución. Se utiliza un bucle while para generar números aleatorios hasta que la probabilidad de que ocurra el número de eventos k sea inferior a un umbral determinado por la función de densidad de Poisson.

Esta implementación es adecuada para este caso porque la distribución de Poisson es una buena elección para modelar el número de operaciones de lectura y escritura que pueden ocurrir en un programa, y la implementación utiliza un algoritmo eficiente y preciso para generar números aleatorios a partir de esta distribución [1]. Además, la implementación permite ajustar fácilmente la tasa lambda y el tamaño del arreglo de números aleatorios generados para adaptarse a diferentes situaciones de prueba .

7. Validación del diseño

Finalmente se puede concluir que el diseño seleccionado cumplió con los requisitos del proyecto resultando en una implementación sencilla y amigable para el usuario, en la figura 6, se puede observar la interfaz final que se creó de la cual se destaca la facilidad para entender las operaciones que el simulador realiza gracias a un buen uso de colores y una tabla con un código de colores para diferenciar los cambios que suceden en ejecución.

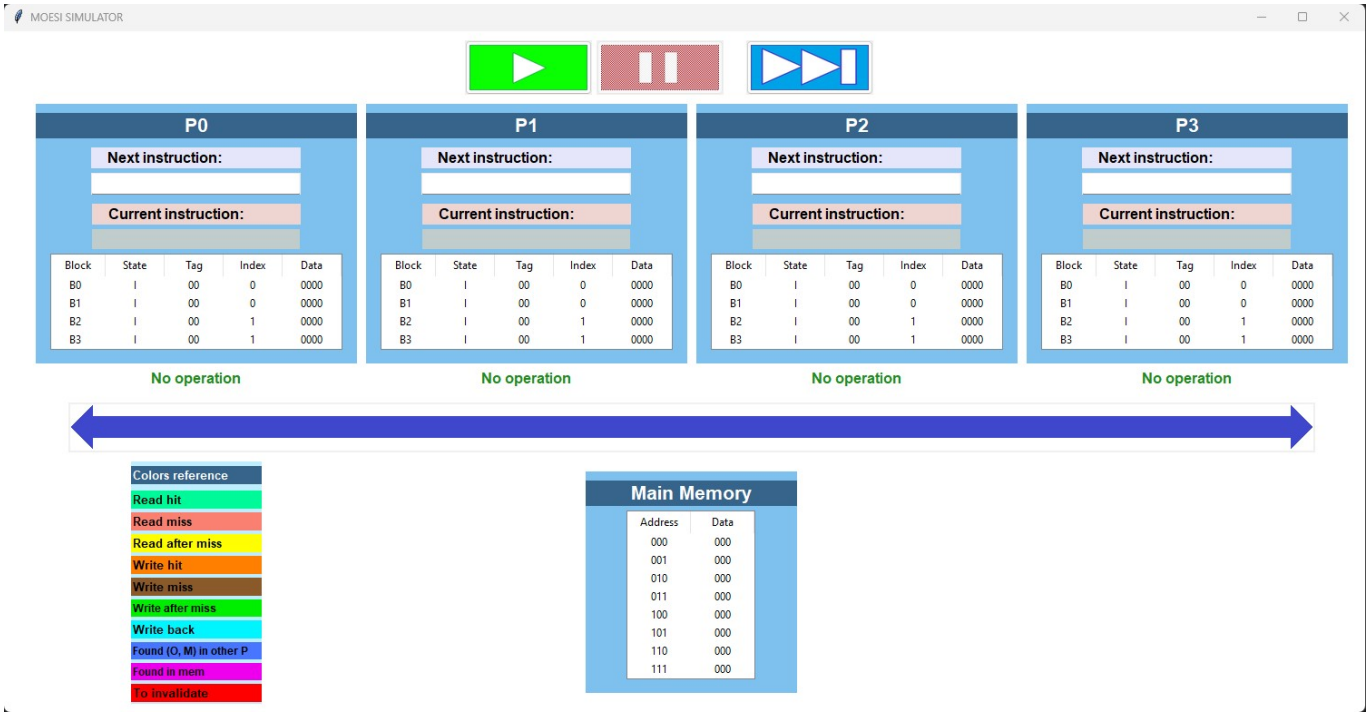

Figura 6: Interfaz resultante de la alternativa seleccionada.

Referencias

[1] MathWorks. "poissrnd - Poisson random numbers."`https://la.mathworks.com/help/stats/poissrnd.html`.