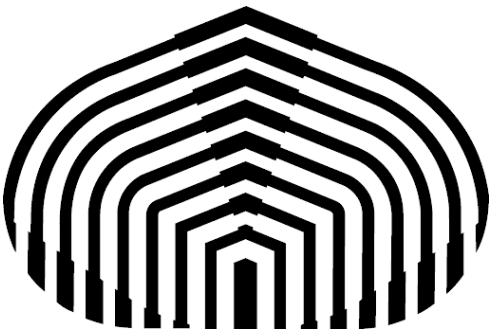

Universidad Simón Bolívar
CI3825 (Sistemas de Operación)
Profesor Fernando Torre

Informe del Proyecto 1

Elaborado por:
Sandibel Soares 17-10614
Juan Cuevas 19-10056

Caracas, Febrero del 2024

Tabla de Contenido

Introducción

Un hilo, también conocido como subproceso ligero, es una unidad de ejecución dentro de un proceso. Es una secuencia de instrucciones que se ejecuta de forma concurrente con otras secuencias de instrucciones dentro del mismo proceso. Los hilos comparten el mismo espacio de direcciones virtual del proceso, pero tienen su propio conjunto de registros y pila.

El presente trabajo tiene como objetivo detallar el desarrollo de un proyecto práctico sobre procesos e hilos en sistemas operativos Unix. El proyecto se enfocó en la creación de un programa que implementa un juego de “Ajedrez Mágico Real” en donde las piezas pueden o no moverse a conciencia propia sin necesidad de recibir una orden por parte del jugador.

Lo que se busca en este trabajo es aprender a crear, gestionar y comunicar varios procesos e hijos de forma eficiente. Al mismo tiempo, queremos gestionar la actividad que estos hilos ejecutan de forma periódica, usando un tiempo calculado, que es la paciencia, para mantener a las piezas en su lugar hasta que estas se les acabe y no tengan otra pieza a la cual cuidar.

Estructura del proyecto

En esta sección nos enfocaremos en explicar brevemente los archivos pertenecientes al proyecto, la utilidad de los mismos. Cabe aclarar que a excepción del archivo main, todos los archivos de extensión (.c) a continuación poseen su respectiva cabecera (.h).

Main.c

Este es el archivo principal del proyecto, que se encarga de unificar el resto de archivos y de ejecutar nuestro programa correctamente luego de compilarlo. Al ser el archivo principal, no tiene funciones adicionales declaradas más allá de la llamada función main que ejecuta el programa.

Este archivo detiene su ejecución solo cuando la partida haya acabado, es decir, cuando hay un ganador o el jugador decide retirarse de la partida.

Board.c

Dicho archivo gestiona el tablero de ajedrez y las acciones que ocurren en él, es decir, también se encarga de gestionar los estados de las piezas al estas moverse de una casilla a otra. De la misma manera, es quien maneja el estado de juego y le avisa al archivo main si el juego acabó.

Cell.c

El archivo cell.c maneja la parte interna de las celdas del tablero, debido que cada celda es 5x5 y las piezas pueden moverse dentro de la misma celda. De la misma manera, gestiona que pieza es dueña de cada celda para así saber que pieza puede moverse a cada celda.

Piece.c

Piece.c contiene la información primordial de las piezas, como la posición en que se encuentran, valor, tipo, color y paciencia de cada pieza, así como el respectivo identificador de cada una. Además, también se encarga de gestionar todos los atributos de las piezas cuando se le solicita.

Menu.c

Se encarga de desplegar un menú con las posibles opciones que puede tomar el jugador en la partida, además de solicitar y verificar las acciones dadas por el jugador.

Pthread_pieces_logic.c

Archivo que se encarga de gestionar todas las solicitudes recibidas por los hilos y actuar en base a ellos para ejecutar acciones sobre las piezas en el tablero. De la misma manera, se comunica con sus hijos para ejecutar (o dejar de hacerlo) alguna tarea.

Child_process_logic.c

Se encarga de gestionar individualmente cada hilo hijo, así mismo, es quien envía y recibe información del padre para ejercer alguna acción sobre las piezas del tablero. Cada hilo hijo es asignado a una pieza, por lo que si esta muere, el hilo dejará de realizar tareas y terminará.

Queue.c

Este archivo se encarga de crear una cola donde almacenar nodos de cierto tipo usando la filosofía FIFO (First In, First Out). En este caso se usará para gestionar la información dada por los hilos hijos al padre en orden de llegada.

Makefile

Archivo que se encargará de compilar todos los archivos del proyecto para crear un ejecutable con nuestro programa. El mismo tiene 2 posibles opciones de uso con make:

- main.out (Por defecto): Compila nuestro proyecto con la versión final del juego de ajedrez.
- clean: Elimina todos los archivos ejecutables creados en la carpeta del proyecto.

Estructuras de datos usadas

Node

Esta estructura tiene dos propiedades, la propiedad value que puede tener la dirección a cualquier tipo y que es el valor que guardará el nodo. Y la propiedad next, que es un apuntador hacia el siguiente nodo creado.

Queue

Queue es una estructura de tipo FIFO, que se encarga de gestionar los nodos del siguiente modo: “El primero en entrar es el primero en salir” como dice el modelo FIFO. Escogimos la cola porque como se podrá ver posteriormente, nos será útil para el manejo de solicitudes en el orden de llegada, lo cual nos será de ayuda para mantener un orden secuencial en las tareas que hay que hacer como el movimiento entre casillas y dentro de las mismas casillas.

PPthreadState

Estructura que contiene la información necesaria para el hilo. Se usa para pasar la información al hilo en el momento de su creación. Además, se

usa para compartir información entre un hilo y su proceso padre. La elegimos debido a que nos ayuda a manejar el estado de cada pieza de forma eficiente, usando la cola para manejar las solicitudes pendientes.

RequestPiece

Esta estructura se escogió principalmente por la facilidad que aporta a la legibilidad del código y evitar el uso de comunicación TLV (Tipo Largo Valor) . Más tomando en cuenta que vamos a pasar en una variable la información de la solicitud del hijo, por lo que es más sencillo de leer y gestionar. Dicha estructura posee 2 enteros, para identificar la pieza y el tipo de movimiento a hacer respectivamente.

RequestFather

La estructura RequestFather fue escogida gracias a que ayuda a gestionar los hilos de las piezas de cada equipo. Tiene como parámetros: 2 enteros que corresponden al identificador de la pieza sobre la que se está trabajando y su acción, y un arreglo de enteros de tamaño 8 con la paciencia de las piezas del equipo. Nos pareció útil esto ya que los dos primeros parámetros sirven para que el padre gestione el hilo actual de la pieza y ejecute las acciones necesarias para ello. Mientras que con el arreglo de enteros sabe qué hilos tiene que mantener en suspenso y a la espera de nuevas órdenes.

InputUtils

Esta estructura está formada por 3 atributos: Un entero que indica el archivo a leer, un apuntador a un hilo y un cola de procesos pendientes. Esta estructura de datos nos fue de ayuda para manejar los movimientos de las piezas a través del tablero, ya que teniendo en cuenta la ruta, ejecuta los

movimientos con una espera de un segundo entre cada uno de ellos siguiendo el orden de entrada en la cola.

PieceType

Es un tipo de dato enum creado para diferenciar al rey de los caballos. Siendo 0 para los caballos y 1 para el rey. Si bien usamos esto para las piezas como un tipo, lo usamos como solo buena práctica para diferenciar de manera clara las piezas.

PieceColor

Tipo de dato enum similar al anterior, usado para diferenciar los colores de las piezas, siendo 0 blanco y 1 negro.

Piece

Estructura con los atributos necesarios para gestionar una pieza. Entre los atributos están un tipo PieceType y PieceColor para identificar tipo y color respectivamente, ocho enteros para el identificador, valor, coordenadas X y Y en el tablero, coordenadas X y Y en la casilla, para verificar si está en movimiento o no (1 se mueve y 0 no) y para llevar la paciencia de la pieza. Además, posee una cola en donde lleva en orden los movimientos que tiene que realizar.

Escogimos esta estructura debido a que nos pareció eficiente la manera de almacenar y gestionar los datos de cada pieza, tomando en cuenta que con el apuntador a un pieza podríamos saber donde está, ubicándola así de manera sencilla en el tablero.

Cell

Cell es una estructura que refleja las casillas en el tablero, en donde va a tener como parámetros una matriz 5x5 y un entero que representa el

identificador de la pieza dueña o que domina esa casilla. Usamos este tipo de estructura porque nos ayuda a definir los límites y movimientos que puede hacer una pieza dentro de la misma casilla, de la misma manera, al asignarle un dueño a la casilla, nos aseguramos que una pieza aliada no tenga como destino una casilla ocupada por su equipo.

Turn

Es un tipo de dato enum creado para diferenciar el turno de juego. Siendo 0 para el jugador y 1 para la computadora (o IA como se llama en el código).

Cursor

Estructura que posee 5 enteros como parámetros, en donde 2 son las coordenadas X y Y del cursor en el tablero, otros 2 son las coordenadas X y Y del cursor en la celda, y el último sirve para identificar si la casilla donde está el cursor es válida para seleccionar. Dicha estructura fue escogida por la sencillez que ofrece a la hora de moverse sobre el tablero y las celdas, además de que nos ayuda a saber de manera eficiente si es elegible una casilla.

Board

Estructura que representa un tablero de ajedrez 8x8, el cual tiene las siguientes propiedades: 3 enteros para los identificadores de lo reyes y el ganador respectivamente; un arreglo de piezas en donde el índice de cada elemento es el identificador que posee la estructura pieza; un arreglo de entero que identifica el tipo de movimiento que hace una pieza (-1 si está quieta, 0 si se mueve por una orden y 1 si se mueve por iniciativa propia) en donde nuevamente, el índice es el identificador de cada pieza; una matriz de 8x8 de celdas que es quien posee las piezas y donde se ejecutan los

movimientos de las mismas; y por último una matriz 33x33 de caracteres usada para imprimir de manera sencilla el tablero tomando en cuenta que las fronteras de una celda las comparte con sus vecinos.

Definimos este tipo de estructura debido a que nos daba las siguientes ventajas:

- Facilidad a la hora de imprimir el tablero gracias a la matriz de caracteres que nos ayuda a unir las fronteras de las celdas en una sola, de la misma manera, nos ayuda a la superposición de elementos en caso de que una pieza pase sobre otra.
- Ayuda a saber de manera sencilla puesto que al tener los IDs de los reyes es fácil saber si alguno fue eliminado durante la partida, por lo que ayuda a finalizar en el momento debido.
- Gracias al arreglo de piezas tenemos de manera rápida y eficiente acceso a las mismas con solo su identificador. Además de que usando apuntadores a dichas piezas sería fácil gestionar los valores de la misma.
- La forma de matriz del tablero ayuda a calcular la distancia de las piezas, usada para la paciencia de una manera más sencilla, tomando en cuenta que en este caso la distancia se calcula usando la distancia Manhattan.

Otros métodos auxiliares

Menú

En el menú usamos una variedad de funciones que que manejan la entrada y salida de las opciones disponibles, entre ellas están:

- ShowMenu: Muestra en la consola el menú de opciones y el carácter necesario para ejercer cada acción.
- enterOptions: Solicita una secuencia de comandos a ejecutar por el juego de un tamaño no mayor a INPUT.

- isValidOption: Verifica si la secuencia de caracteres de tamaño definido INPUT es una secuencia válida. Por lo que, la secuencia debe tener solo caracteres correspondientes a las acciones disponibles.
- willExit: Verifica si el usuario quiere salir, es decir, revisa si el usuario escribió el caracter q en la secuencia de caracteres leída. Además, detiene los subprocesos que se estén ejecutando.

Cabe acotar que definimos la variable INPUT como un valor constante el cual puede ser usado en el resto del proyecto.

Función principal

La función principal del proyecto lo que hacemos es inicializar el tablero y los pipes que van a comunicar al padre con los hijos. Además, inicializamos el proceso hijo en conjunto con sus hilos necesarios para operar las piezas y la variable con las coordenadas del cursor.

Posteriormente, tenemos un ciclo el cual va a ejecutarse hasta que tengamos un ganador (Sea la computadora o el jugador), y mientras el mismo se esté ejecutando, mostrará el tablero y el menú de opciones, además de pedir la secuencia de acciones que el jugador quiere realizar, entre las opciones, está la condición de que si el jugador quiere salir, el juego se detiene. En caso contrario sigue y ejecuta las acciones insertadas por el usuario.

Después de eso, ejercemos la acción de mover el cursor (Si no ingresa ningún movimiento el cursor se mantiene estático), revisamos que en caso de haber una selección y/o jugada, estas sean correctas. Luego, en base a las acciones que haya hecho el usuario, y ejecuta mediante hilos las acciones de mover la pieza, para luego repetir el ciclo.

Finalmente, en caso de que el juego haya terminado, se muestra al ganador.

Conclusiones

En este proyecto, se desarrolló un programa en lenguaje C que implementa el juego de “Ajedrez Mágico Real”. El programa demostró la capacidad de gestionar los hilos para que las piezas se puedan mover de manera independiente. Se analizaron las ventajas y desventajas de la programación con procesos e hilos, y se identificaron algunas áreas de mejora para futuras investigaciones.

Durante el desarrollo del proyecto, se encontraron algunos desafíos relacionados con la sincronización de los hilos y la gestión de recursos. Se aprendieron técnicas y herramientas útiles para la programación concurrente, como mutex y semáforos.

Se recomienda que en caso de seguir con este proyecto, hacer énfasis en la gestión de hilos para la toma de decisiones de las piezas, ya que esto puede traer algunos inconvenientes a futuro. Así mismo, pese a ser una versión distinta al ajedrez tradicional, se logró implementar de cierta forma como si fuera un juego de ajedrez tradicional, lo cual facilita un poco la implementación del juego.

Para finalizar, se recomienda investigar más sobre otros métodos de gestión de hilos y procesos con memoria compartida como sería el mapeo de discos, los cuales podrían ser útiles para este tipo de proyectos.

Agradecemos a nuestro profesor y compañeros por apoyarnos durante este proyecto.