
República Bolivariana de Venezuela

Ministerio del Poder Popular para la

Universidad Simón Bolívar

Departamento de Computación y Tecnologías de la información


CI-3825: Sistemas de Operación 1

# PROYECTO 1: AJEDREZ MAGICO REAL


Integrantes:

Luis Isea  19-10175

Jesús Prieto 19-10211


23 de febrero del 2024

# INTRODUCCIÓN

En el vasto universo de Harry Potter, la magia se entrelaza con el aprendizaje, la aventura y la estrategia en cada rincón de Hogwarts. Dentro de sus muros centenarios, se encuentra un juego que va más allá de las simples jugadas sobre un tablero: el Ajedrez Mágico de Hogwarts

En este juego, la estrategia va más allá de mover piezas sobre un tablero; implica comprender las complejidades de las criaturas mágicas que pueblan el tablero y anticipar sus movimientos impredecibles, las piezas están encantadas para tener algún grado de autonomía, lo que agrega un elemento de imprevisibilidad y emoción al juego. Esto desafía a estudiantes y magos por igual con sus intrincadas estrategias y sorpresas inesperadas.

Debido a su nivel de imprevisibilidad se le ha encomendado a los estudiantes del curso de Sistemas de Operación I como proyecto realizar un programa sobre El Ajedrez Mágico de Hogwarts, usando como lenguaje de programación C.

Al presentar una complejidad única que requiere un enfoque meticuloso en la gestión de recursos y la coordinación de acciones entre las distintas entidades del juego lo vuelve un desafío perfecto para los estudiantes y sobre la materia, abarcando con esto el área de los procesos, la utilización y creación de hilos dentro de los procesos para gestionar diferentes áreas del un programa, en este caso el movimiento y la interacción de las piezas del tablero, y por último la visualización de cómo se comunican un proceso con otro mediante las distintas formas , ya sea por pipe o señales.

Debido a una mala administración del tiempo no se logró completar al máximo la tarea encomendada. Por lo tanto en este informe, detallaremos el diseño de los componentes implementados y las ideas que se tenían para resolver las partes faltantes del programa, además de esto también se dirá los desafíos encontrados durante el proceso y las soluciones que hemos desarrollado para superarlo.

## DISEÑO DEL PROGRAMA

### Representación del tablero

Para representar el tablero del juego se hizo uso de una matriz 34 x 68, en un principio se tuvo a la idea de usar una lista enlazada para guardar el tablero pero esto generaba errores al momento de realizar la impresión del tablero y de mover un elemento a través de la lista, lo cual era necesario para la idea del cursor del jugador. La primera y última fila son para cerrar el tablero, por lo cual no se toma en cuenta al momento de mover una pieza, de igual esto se aplica a las primeras 3 columnas y a la última.

La matriz se generaba por medio de una función ubicada en el archivo chessboard.h, esta función nos retorna un apuntador a la matriz creada. Se decidió que cada casilla en el tablero tuviera el tamaño de 4x7, esto por nivel estético, pues hacerla de 5x4 hacía que se viera el tablero de forma estirada. Con esta decisión se produjo un aumento en la cantidad de filas y columnas debe recorrer en la matriz para llegar de una casilla a otra.

### Representación de las piezas

Para ambas piezas tanto la del jugador como las del enemigo se decidió realizar una función que nos devolviera un arreglo de struct que contuviera, los siguientes atributos:

- enum status: indica si está o no en el tablero
- enum type: indica si es caballo o rey.
- int x: indica la posición de la pieza en x.
- int y: indica la posición de la pieza en y.
- int casillaX: indica la posición de la casilla en x.
- int casillaY: indica la posición de la casilla en y.
- enum color: indica si la pieza es blanca o negra.

- char symbol: indica qué representación tiene la pieza (N, z para los caballos y K,x para los reyes).

De esta forma cada struct sería una pieza del tablero y tendría sus propios atributos. Se crearon dos funciones que lograran pero ambas usan el struct explicado anteriormente. Una función es para crear las piezas del jugador y la otra para las piezas del enemigo o de la computadora.

**Representación de los hilos**

Al momento de crear un hilo, se nos pide como parámetro un apuntador hacia el argumento para la función de rutina, ahora bien esta función puede o no necesitarla, ya depende del criterio del programador incluirla o no para su rutina. En nuestro caso si fue necesario para poder describir las rutinas de movimientos para el caballo, el rey y determinar la casilla en donde se encuentra una pieza luego de terminar un movimiento.

El problema se presenta en que una rutina solo permite el ingreso de un parámetro, si es necesario dos o más parámetros para la rutina se nos genera un problema. Para resolver esto se usó un struct para almacenar los atributos necesarios para cada pieza al momento de realizar la rutina. Estos atributos son:

- char ** chessboard: tablero de ajedrez.
- struct piece *piece: arreglo de piezas del jugador o del enemigo.
- int pipe_1[2]: arreglo de 2 cifras, se usará para el pipe.
- int pipe_2[2]: arreglo de 2 cifras, se usará para el pipe.
- int id: identificador de la pieza e hilo trabajado.
- int paciencia: paciencia de espera de la pieza.
- pthread_t hilo: hilo contenida en la pieza, esto servirá para iniciar los hilos.

**Estructura y comunicación de procesos**

Para trabajar de forma independiente y paralela las distintas piezas y el cursor del juego se ideó hacer 3 procesos hijos, uno para las piezas del jugador, uno para las piezas del enemigo y otro para el cursor del juego. Como se dijo antes, la mala planificación del equipo produjo que se lograra solo la implementación de 2 procesos, el de las piezas.

El proceso padre tenía la tarea de esperar que terminara cada proceso hijo, con cada nuevo proceso hijo se debía crear un arreglo de hilos con la estructura definida anteriormente, esto se lograba usando la funciones de la librería pthread, luego era necesario inicializar los pipe o tuberías definidas para cada hilo, pues esta nos permitirá enviarle los cambios hechos durante el proceso al padre.

Una vez terminada la activación de los pipe se procedía a iniciar dos rutinas para el hilo indicado, una rutina realizaba el movimiento y la otra determinaba la nueva ubicación de la casilla(casilla no es lo mismo que posición en la matriz del tablero) y allí terminaba el proceso. Tanto el primero proceso hijo como el segundo realizan todos estos pasos.

Mientras tanto el padre debe esperar que termine el primer hijo, cuando éste culmina procede a leer los pipe de cada hilo perteneciente al arreglo de hilos, así actualiza su estado. Una vez que se terminan de leer los pipes realiza una primera impresión del tablero actualizado y procede a esperar al segundo hijo, repitiendo los pasos que el primer hijo.

**Señales durante los procesos**

Se crearon 3 manejadores de señales, dos para los casos en donde alguna pieza se comía a otra y una para cuando se comian a un pieza rey. De esa forma el jugador podía ver un mensaje en la consola que le indicará si ocurre algún suceso que pasará desapercibido.

## DIFICULTADES ENCONTRADAS

El equipo se encontró principalmente con los siguientes desafíos:

**Dificultades generales**

- Sistema de comunicación entre procesos: Decidir cómo se comunicara un proceso padre con su hijo y viceversa era un tema de vital importancia, de las distintas formas que había debíamos escoger las más adapta para nuestra estructura, además que debía permitir que la comunicación entre procesos fuera eficiente, con la menor cantidad de errores y con una sincronización.
- Estructura de los hilos y sus funciones de rutina: Como definir la estructura del hilo fue otro problema para nosotros, la incapacidad de indicar mas parametros en la rutina nos obligó a decidir qué atributos serán necesarios en cada rutina.
- Actualización con los pipes: Hubo cierta confusión al momento de manejar los pipes, varias ocasiones no se realizaba la acción de escritura el pipe por una mala indicación de quien se debía cerrar.
- Estructura para los procesos: Fue complicado decidir cuántos procesos usar para distribuir completamente la carga del programa. Y asegurarse que ningún proceso ejecutará la acciones que no le corresponden, sino a otro.

**Decisiones Tomadas**

- Uso de matriz para representar el tablero.
- Uso de struct para definir las piezas y los hilos.
- Utilizar un proceso encargado de mover las piezas del jugador y un proceso para que moviera las piezas de la computadora. Ambos procesos tienen un proceso hijo que maneja los hilos correspondientes.

# CONCLUSIONES

## Estado actual del programa

Debido a la falta de organización en el tiempo de entrega, no se pudo terminar una versión del todo jugable del Ajedrez mágico Real. Esto afectó en la creación del sistema de I/O para que el jugador pueda moverse y seleccionar piezas, en la creación de la función completa que valide si una orden es válida(no contemplamos todos los requisitos para que sea válida), el sistema de contraordenes para las piezas, la impresión por segundos de las piezas y por último el sistema de "conciencia" para las piezas.

Lo que logró nuestro grupo fue crear varios procesos para cada pieza, crear los hilos, mostrar el tablero, mover las piezas, manejar el concepto de turnos y terminar el juego. Se tuvo que colocar una cota de máximo 100 turnos para cada partida en caso de que no se llegue a comer el rey. Esto porque la partida no termina hasta que un jugador capture al rey del contrincante, y como las piezas pueden moverse solas, a veces puede ser difícil que esto pase.

## Lecciones aprendidas

El desarrollo de este programa permitió poner en práctica conceptos clave:

1. Diseño y estructuración del programa: Es importante concebir desde un principio la estructura de datos que se maneja en el programa, sin embargo, hay que estar abiertos a la idea de cambiar esta estructura durante el desarrollo del programa, ya que se pueden encontrar estrategias más fáciles de implementar o que sean más eficientes. En este proyecto se tuvo que replantear más de una vez la estructura de datos.
2. Comunicación entre procesos.
3. Gestión de hilos.
4. Sincronización entre procesos e hilos.