Práctica

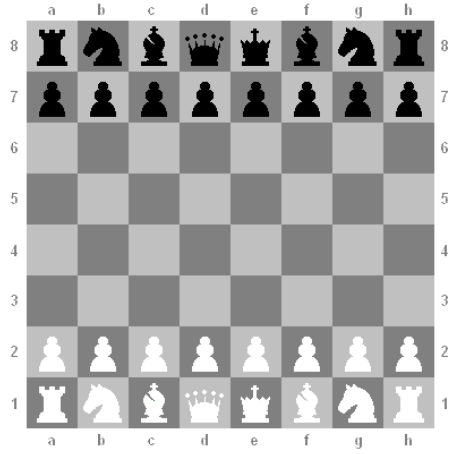
3

Sistemas Operativos

Threads: creación, sincronización y comunicación

Enunciado

La práctica tiene el objetivo de utilizar hilos sincronizados para simular un torneo de ajedrez en el que participan numerosos jugadores simultáneamente.



Consideraremos una variante del juego simplificada, que se desarrolla sobre un tablero de 8x8 casillas, y en la posición inicial cada jugador (blancas y negras) cuenta con 16 piezas, ubicadas en las dos primeras filas de cada bando. Todas las piezas son iguales (como damas), y comienza el juego el conductor de las piezas blancas. Cuando es el turno de un jugador, selecciona aleatoriamente una de sus piezas y elije igualmente al azar el destino: en el caso de que la casilla este vacía, se completa el movimiento; si está ocupada por una pieza propia, no se realiza el movimiento (pasa su turno) y si se trata de una pieza del rival, se la "come" y se retira del tablero.
Vence un jugador cuando antes de la jugada J (>120), ha comido todas las piezas rivales, recibiendo 2 puntos (y el perdedor 0). En caso de no haber resuelto la partida antes del movimiento J, es declarada como tablas (1 punto para cada jugador).

Al finalizar una partida, se actualizan 3 cuentas para cada jugador: el nº de movimientos realizados (`NUM_MOV`), el nº de piezas capturadas (`NUM_CAP`) y el nº de puntos obtenidos (`NUM_PTS`).
Todas las constantes que identifiques deben figurar en un fichero denominado "`constantes.h`".

1. Simular un match que se disputa entre dos jugadores, como el mejor después de jugar 100 partidas. En este caso, cada jugador se modela mediante un thread que espera su turno para jugar, de acuerdo a las reglas anteriores. Cada jugador recibe un identificador (`ID`, nº entero) que es diferente para cada uno de ellos y el color con el que juegan (blanco o negro codificado como un char 'b' / 'n'). El tablero (`TAB`) y los 3 contadores `NUM_MOV`, `NUM_CAP` y `NUM_PTS` son estructuras de datos compartidas por ambos threads. Tienes que diseñar esas estructuras de datos.
   Cada thread, al terminar una partida, actualiza los contadores `NUM_MOV` y `NUM_CAP`, devolviendo el valor de `NUM_PTS` al thread principal, que lleva la cuenta de victorias de cada jugador, así como el número de partidas jugadas. Cuando una partida finaliza, imprime una línea con los valores:
   ```
   Nº Partida, ID Blanco, ID Negro, Ganador (B/N/=), Número movimientos
   ```
   Al final del encuentro, se imprime la información de número de partidas jugadas (NP), las victorias del blanco (B), las del negro (N) y las tablas (T), así como los valores promedio de los movimientos realizados (PM) y de las capturas que han tenido lugar (PC). Ejemplo:
   ```
   NP:100, B:42, N:30, T:28, PM:58.3, PC:14.2
   ```
2. Parte opcional. El apartado anterior nos ha permitido modelar un match entre dos jugadores. Basándonos en ese desarrollo, se requiere simular el desarrollo de `M` matches **simultáneamente**. Hay dos equipos (A y B) que cada uno de ellos tiene `M` jugadores. El valor de `M` se recibe por la línea de comandos (`argv`). Las estructuras de datos `NUM_MOV`, `NUM_CAP` y `NUM_PTS` son compartidas por todos los threads activos, mientras que se creará un tablero `TAB` por cada encuentro entre dos jugadores. Una vez que cada encuentro ha finalizado (recuerda que son 100 entre cada par de jugadores), se imprime una línea con los mismos datos que en apartado anterior, añadiendo al final un resumen que agrupe los resultados e indique el bando ganador. Ejemplo:
   ```
   Equipo ganador: A (M: 30)
   P:3000, B:1470, N:1065, T:465, PM:78.4, PC:14.7
   ```

Realización

A tener en cuenta:

1. Todo el trabajo debe estar en un directorio para esta práctica, denominado "`practica_3`". Debes utilizar las facilidades de sincronización entre hilos para garantizar el correcto funcionamiento del programa y evitar accesos no controlados a los datos compartidos.
2. Debes construir un fichero Makefile que facilite la compilación del programa
3. Debes construir una macro que lance tu programa varias veces, modificando en cada ejecución el número M para la simulación efectuada (empezando en 10 y acabando en 100, incrementando en 5 en 5). El resultado se debe ir añadiendo al fichero "`resultados.txt`".
4. Cuando lo tengas debes entregar un único fichero de formato `tar` con lo que has realizado (explicación en comentarios, fuentes, datos, constantes, `make`, entrada, macro, …) a través de Moodle. Se te valorará la calidad del código, que todo funcione y la macro y `make` realizados, así como la documentación / comentarios y el fichero de resultados final "`resultados.txt`".

Ayuda

1. El tablero se puede implementar como una matriz de 8x8 (o un array de 64) de enteros con los siguientes posibles valores: 0: cuadro vacío, 1 pieza blanca, 2 pieza negra.
2. Las piezas de cada color, al ser iguales, no hay que diferenciarlas de ninguna manera. Debes controlar cuando una pieza es comida por el contrario a la hora de realizar el movimiento, donde estaba (para quitarla) y donde va a estar (para ponerla). Para facilitar el trabajo de localizar las piezas de cada bando cuando quieras elegir una pieza a mover, además del tablero puedes tener dos listas de piezas (una para cada color) que contenga información sobre la casilla en la que están situadas.
3. Se tienen que sincronizar los hilos convenientemente, de tal manera que las variables compartidas deben estar protegidas por mutex.
4. Piensa también como se puede implementar los turnos. Ya que los jugadores se deben ir alternando. Pista: variables condicionales.