Fundamentos de Redes. Memoria de la práctica 2

José María Gómez García
Manuel Herrera Ojea

Para los ejercicios 1 a 4 la forma de proceder ha sido similar. Al comprender qué nos estaba pidiendo cada ejercicio, acudimos a la documentación aportada por los profesores de la asignatura para conocer los métodos de las bibliotecas que teníamos que utilizar para desempeñar las tareas requeridas. En cada uno de los tres primeros ejercicios especificamos con comentarios, dentro del código, las modificaciones que se han realizado sobre el anterior para añadir las funcionalidades necesarias en cada punto, mientras que para el cuarto partimos de un esqueleto diferente.

En los ejercicios 1 y 2 pudimos proceder sin problemas.

El ejercicio 3 sí conllevó algún problema, pues tuvimos que diseñar un método nuevo, run(), para la extensión de la clase Thread y conseguir así el paralelismo a nivel de hebra necesario en este apartado.

El ejercicio 4 supuso más problemas por tratarse de un cambio más brusco de paradigma que con los anteriores. No nos pareció trivial comprender la forma necesaria de proceder a la hora de utilizar los métodos de la biblioteca proporcionada, pero, tras varias preguntas al profesor y haber alcanzado un mejor esquema mental de qué efecto causan los métodos que estábamos utilizando, conseguimos hacerlo con claridad.

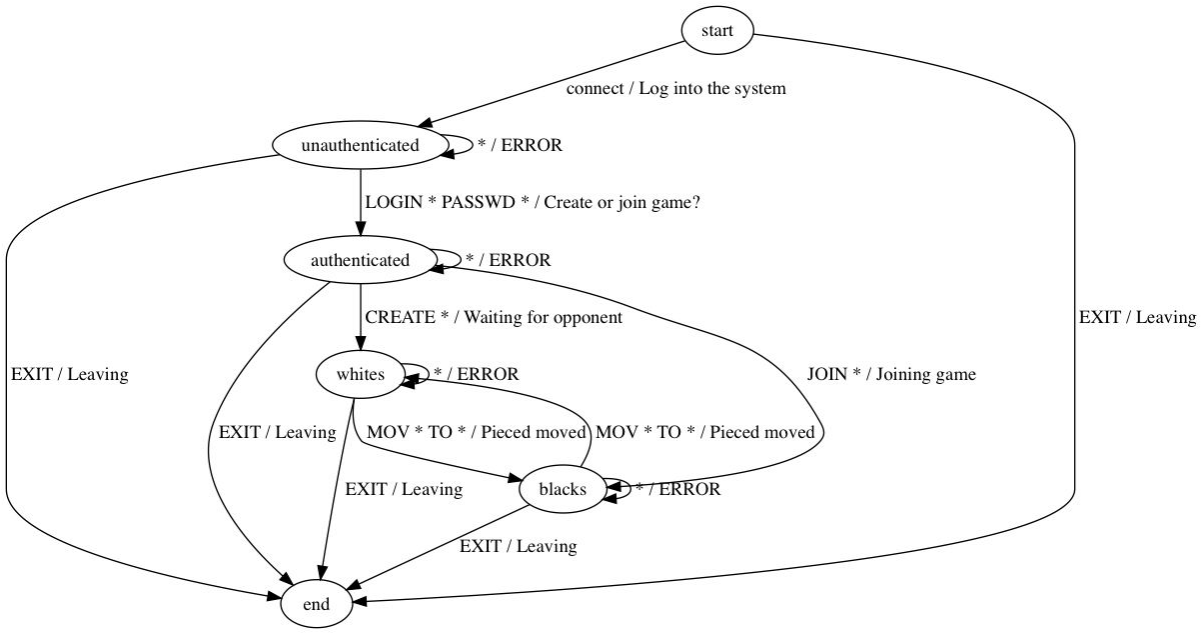
Para el ejercicio 5 hemos utilizado una versión del juego de ajedrez que habíamos diseñado en Java anteriormente, y hemos implementado un servidor en el que parejas de jugadores puedan jugar a través del servidor.
Nos hemos basado para ello en el programa que realizamos en el ejercicio 3, eliminando la clase Procesador.

El servidor se encarga de atender clientes entrantes y de crear una nueva hebra para que los atienda. El servicio que crea entonces es quien gestiona las peticiones del cliente, de forma concurrente. Este proceso y la separación entre servidor propiamente dicho y servicio es completamente transparente para los clientes, bajo cuyo punto de vista están simplemente enviándole peticiones a un único servidor.

En la clase del servicio se almacenan de forma estática en un vector los juegos que hay en cada momento en el servidor. Cada proceso concurrente del servicio utiliza el mismo vector para comunicarse indirectamente entre ellos y consultar el estado de la partida para actualizar la información que se le proporciona al cliente.

No ha sido necesario el uso de semáforos, pues en ningún momento va a haber dos procesos intentando escribir a la vez en el vector de juegos activos. Por el protocolo que hemos establecido, sólo dos procesos como máximo pueden estar a la vez utilizando uno de los juegos del vector, y, siempre que uno pueda escribir en él un nuevo movimiento, el otro solo podrá leer el estado actual del tablero, por lo que no habrá interferencias indeseadas.

Este es el diagrama de estados en el que nos hemos basado para la implementación del servidor:



Para establecer conexión con el servidor es necesario que el cliente envíe al servidor el mensaje ‘connect’. Tras esto es necesario iniciar sesión, siguiendo la sintaxis ‘LOGIN <nombre de usuario> PASSWD <contraseña>. Por defecto en el código del servidor está definido el usuario ‘A’ , con contraseña ‘A’.

A continuación el cliente tiene la opción de crear un juego o unirse a este. Para crear un juego es necesario mandar el mensaje ‘CREATE <code>’, siendo <code> un número del 1 al 10 y no pudiendo haber un juego creado con el mismo código. Este número será el código del juego; el creador del juego será por defecto el color blanco.
A la hora de unirse es necesario mandar el mensaje ‘JOIN <code>’, siendo <code> un código de juego ya existente; al jugador que se une a un juego ya existente se le asigna el color negro.
Esto se gestiona mediante un vector de valores booleanos, donde se almacena si un jugador ya ha comenzado un juego al que pueda unirse otro,

Una vez creado el juego las blancas pueden empezar su movimiento, enviando el mensaje ‘MOV <columna_ini><fila_ini> TO <columna_dest><fila_dest>’, siendo <columna_ini> y

<fila_ini> la columna y la fila de la pieza que se desea mover, y <columna_dest> y <fila_dest> la columna y fila de la casilla a la que se quiere mover la pieza.
La aplicación del ajedrez es quien se encarga de gestionar que un movimiento sea o no válido, siendo esta una responsabilidad que no recae sobre el protocolo de comunicación que hemos definido.

Una vez que se llega al jaque mate la partida finaliza y anuncia el ganador. El usuario puede entonces crear de nuevo un juego o unirse a otro ya existente, de la misma forma que justo después de haberse identificado en el sistema.

Además, en cualquier momento durante su conexión el jugador puede finalizar la sesión enviando el mensaje ‘EXIT’ al servidor.