Trabajo Práctico: Sistemas Operativos

Mellino, Natalia Farizano, Juan Ignacio

Objetivo

El objetivo es implementar un servidor distribuido de partidas de ta-te-ti. Este le permite a los usuarios conectarse, crear nuevas partidas y aceptarlas, así como unirse a una partida iniciada o por iniciar como observador. Las distintas opciones entre las que puede elegir el usuario, son las siguientes:

1. Iniciar la comunicación con el servidor: para esto el usuario provee un nombre y si este no se encuentra en uso el servidor lo acepta.
2. Listar los juegos disponibles: se muestran tanto los juegos que están en desarrollo como los que están esperando un contrincante.
3. Crear un nuevo juego: este será entre el jugador que lo cree y el primero que lo acepte.
4. Realizar una jugada: el usuario puede realizar un movimiento en el tablero o abandonar la partida si lo desea. El servidor devolverá un error si la jugada no es válida.
5. Observar un juego: el usuario que pida observar un juego, comenzará a recibir todos los cambios en el estado del juego.
6. Dejar de observar un juego.
7. Terminar la conexión: el usuario abandona todos los juegos en los que participe u observe y luego la conexión se termina.

Al ejecutar el programa del cliente, este tendrá un menú interactivo con instrucciones detalladas acerca de cómo realizar cada comando.

Ejecución del Cliente y Servidor

Para ejecutar el Cliente, en la consola de Erlang debemos hacer primero `c(client).` y luego, `client:start(IP, Port).` donde IP y Port deben ser la direccion de IP y Puerto del servidor al que se desea conectar.

Para ejecutar el Server, debemos cargar dos módulos con: `c(tateti).` y `c(server).`, una vez hecho esto, simplemente escribimos: `server:start().` y ya estará listo para aceptar conexiones e interactuar con los clientes.

Observaciones:

- Para ejecutar el servidor se asume que los nodos en las PCs ya están conectados entre sí.
- Por la manera en que se intercambian los mensajes entre Cliente y Servidor en nuestra implementación, es importante aclarar que para el funcionamiento correcto del Servidor, el Cliente debe estar programado en Erlang.

Implementación

Servidor

Cuando se incializa el servidor se abre el socket y comienzan cuatro procesos:

- `dispatcher`: es el proceso encargado de aceptar las conexiones de los clientes entrantes. Al aceptar la conexión de un cliente, se inicia un proceso llamado `pupdater` que se encargará de mandarle al cliente las respuestas a sus peticiones y las actualizaciones acerca del estado de los juegos en los que participe. Luego, el `dispatcher` inicia un hilo `psocket` que es el que se encargará únicamente de atender los pedidos de dicho cliente.

  El hilo `psocket`, primero se encarga de que el cliente proporcione un nombre de usuario. Una vez que este ya tiene un nombre, puede comenzar a realizar pedidos. Cuando el cliente hace un pedido, se inicia el proceso `pcomando` que recibe el pedido que mandó el cliente y si no se produce ningún error, lo cumple y responde a la petición del cliente.

- `master`: este proceso es el que guarda la información de los jugadores y los juegos que están en curso. Cuando el proceso `pcomando` recibe la petición del cliente, este se la manda a `master`, que se encarga de realizarla, actualizar su información (si es necesario) y luego responderle a `pcomando` con la información que éste le debe enviar de vuelta al cliente y si es necesario, los cambios que deben informarse a los demás jugadores y/u observadores.

- `pbalance` y `pstat`: como podemos tener distintos servidores corriendo en distintos nodos, aparentando ser un único servidor, es necesario tener funciones que se encarguen de repartir el trabajo que debe realizar cada nodo, para asi evitar que en un solo nodo se realicen muchas peticiones mientras que los otros no están trabajando. Para ello, el proceso `pstat` manda regularmente la información de carga del nodo en el que se encuentra al resto de los nodos. Esta información la recibe `pbalance` y se encarga de guardarla. Entonces, cuando un cliente hace una request, se le pide a `pbalance` que nos devuelva el nodo que menos carga tiene en ese momento, para así realizar la petición del cliente en dicho nodo.

En resumen, son estos procesos, junto con algunas funciones auxilares, las que conforman y hacen funcionar nuestro servidor, estas tratan de ser lo más exhaustivas posibles al momento de detectar si hubo algún error en la conexión o al procesar una petición para poder evitar que éste se caiga.

Cliente

Cuando inicializamos el Cliente con la función `start`, ésta comprueba primero si no hubo ningún error al intentar conectarse. Si la conexión fue exitosa, se llama a la función `sender`, que se encarga de que el usuario proporcione un nombre. Una vez que ya lo tiene, se inicia el proceso `receiver` y `sender` se vuelve a llamar recursivamente. Ya en esta parte, el `sender` se encarga de enviar al Server todas las peticiones del usuario. Por otro lado, el `receiver` es el que recibe las respuestas por parte del Server y se encarga de mostrárselas al usuario por pantalla.

Básicamente, la implementación del Cliente consta de un menú interactivo, donde al usuario se le presentan una serie de opciones para elegir, y cuando elige alguna de ellas, estas son convertidas a peticiones que luego serán mandadas por el `sender` al Server a través del Socket y el `receiver` se encargará de comunicarle la respuesta del Servidor al usuario.

Tateti

El módulo tateti, es tan sólo una implementación simple y práctica del juego tateti, podemos ver que la función principal es `make_play`, la cual se encarga de realizar la jugada pedida por el usuario y modificar el tablero, siempre controlando que los movimientos que se realicen sean válidos.