Sistemas Operativos

Trabajo Práctico 2: Erlang

Juan Cruz de la Torre, Bautista Marelli

Junio 2020

1. Modo de Uso

Para iniciar el servidor, se puede usar la función `server:start/0`. Por defecto, crea un Socket TCP que escucha en el puerto 8000, pero si ya está siendo utilizado se busca un puerto disponible distinto. Si se quiere hacer uso de la funcionalidad distribuida del servidor, puede ejecutar la misma función en todos los nodos que se desee mientras que esten conectados entre sí.

Para iniciar el cliente, se puede usar la función `client:start/1`, que toma como argumento el puerto donde esta corriendo el servidor. Vale aclarar que la IP predeterminada a la que se conecta el cliente es la local (`127.0.0.1`).

2. Implementación

Server

La implementación del servidor sigue las pautas establecidas por la descripción del Trabajo Práctico, y consta de varios procesos que se describen a continuación.

- `dispatcher`: Es el proceso que se encarga de aceptar clientes que quieren conectarse al servidor, y de crear un proceso `psocket` asociado a cada nuevo cliente.
- `userlist`: Es el proceso que se encarga de llevar el registro de todos los clientes registrados en el servidor. Cuando un cliente se conecta, debe registrarse en esta lista usando el comando `CON NombreDeUsuario`. Dicho nombre de usuario no puede contener los caracteres `|` ni `,` ya que estos impiden el funcionamiento correcto de varias operaciones de parseo. La manera en que se representa la lista de usuarios es a través de un map cuyas claves son los nombres de usuario y los valores asociados son los identificadores del proceso `psocket` asociado a cada uno.
- `gamelist`: Es el proceso que se encarga de llevar el registro de todos los juegos registrados en el servidor. La manera en que se representa la lista de juegos es a traves de un map cuyas claves son los nombre de usuario y los valores asociados son records `game (player1, player2, board, turn, observers)`. Se encarga, además, de procesar y responder acordemente a los comandos ejecutados por los clientes sobre los juegos activos.
- `pstart` y `pbalance`: El proceso `pstart` es el que se encarga de mandar a todos los nodos la actualización de las cargas del nodo en el que se encuentra (calculadas simplemente usando `erlang:statistics(run_queue)`. El encargado de recibir estas actualizaciones es el `pbalance`, quien lleva una lista actualizada de las cargas de los nodos para poder elegir el de menor carga cuando un `psocket` requiera crear un `pcomando`.
- `psocket`: Es el proceso que cada cliente tiene asociado en el servidor, se encarga de recibir los mensajes del cliente, enviárselos a un `pcomando` en el nodo con menor carga (según lo establecido en `pbalance`) para ser procesados, recibir la respuesta de los `pcomando` creados y responderle acordemente al cliente. Debido a que todo se realiza de manera asíncrona, se hace uso del modo {`active, once`} de los Sockets TCP de Erlang. De esta manera, el proceso psocket puede recibir tanto mensajes del cliente como de otros procesos de manera sencilla.

- `pcomando`: Es el proceso que se crea cada vez que un `psocket` recibe un mensaje de un cliente, creado en el nodo con menor carga (según lo establecido en `pbalance`), y que se encarga de parsear el mensaje del cliente, comunicarse con los distintos servicios y nodos según el comando recibido, y comunicarse nuevamente con el `psocket` una vez obtenida una respuesta.

Algunas consideraciones sobre los distintos comandos permitidos:

- `LSG`: el formato en que se envía la lista de juegos al cliente lleva el formato `OK CMDID Juego1 Juego2 ...`, en donde `JuegoN` se representa como una cadena dada por `GameId,Player1,Player2` o `GameId,Player1,none` si nadie aceptó el juego todavía.

- `PLA` y `OBS`: cuando un jugador realiza una jugada en una partida, se le envía como respuesta `OK CMDID GameId board BoardStr`, donde `BoardStr` es una representación en cadena del tablero actual, o `OK CMDID GameId game_ended Winner` si el juego ha terminado, donde `Winner` es el nombre del jugador que ganó o `DRAW` si hubo un empate. Cuando un cliente se encuentra observando un juego, recibe actualizaciones del servidor de manera similar pero con prefijos `UPD` en lugar de `OK`.

- `server:display_current_state/0`: si se desea, puede ejecutarse esta función durante la ejecución del servidor para ver una lista de usuarios registrados, juegos activos, y que juegos cada usuario está jugando y/o observando en un momento dado.

De todas las combinaciones de comandos que pueden de mandar los clientes, muchas de esas son invalidas y devuelven un mensaje de error. Tal mensaje es de la forma `ERROR CMDID error_code ...` o `ERROR error_code ...` (si no tiene sentido hablar de un `CMDID`). A continuación se describen los posibles mensajes de error emitidos, junto con una explicación breve de los mismos:

- `invalid_username`: si el nombre de usuario contiene `|` o `,`

- `username_taken`: si ya existe un usuario en el mismo nodo con ese nombre de usuario

- `not_registered`: si el cliente intenta realizar un comando antes de registrarse

- `already_registered`: si el cliente intenta volver a registrarse

- `invalid_game_id`: si el id del juego es invalido

- `already_playing`: si el cliente intenta acceder a un juego que ya está jugando

- `game_full`: si el cliente intenta acceder a un juego que ya tiene 2 jugadores

- `cant_observe_self`: si el cliente intenta observar un juego el cual el ya está jugando

- `already_observing`: si el cliente intenta observar un juego el cual el ya está observando

- `not_an_observer`: si el cliente intenta dejar de observar un juego que no está observando

- `not_a_player`: si ek cliente intenta realizar una jugada en un juego que no está jugando
- `wait_for_opponent_to_join`: si el cliente intenta realizar una jugada en un juego que solo hay un jugador
- `not_your_turn`: si el cliente intenta realizar una jugada en un juego cuando no era su turno
- `invalid_move`: si el cliente intenta realizar una jugada invalida en un juego

Client

Realizamos una implementación simple de un cliente, la cual no es para nada exhaustiva de la funcionalidad ofrecida por el servidor, y consta de las siguientes partes:

- `start`: Conecta el cliente con el servidor, según el puerto que recibe como argumento. Este comprueba que la conexión sea exitosa y pide al cliente que se registre. Una vez realizada la registración, se lanza el proceso `receiver` y se ejecuta la función `sender`.
- `sender`: Es la función que se encarga de recibir por entrada estandar comandos de parte del usuario, y enviarlos al servidor. Una lista de comandos disponibles se muestra al usuario si este ingresa `HELP`.
- `receiver`: Es la función que se encarga de recibir los mensajes enviados por el servidor.