TP2 S.O.

Agustín Gurvich, Santiago Wirzt

29/06/2020

1 Estructura de módulos y procesos

La implementación del servidor cuenta con tres módulos, uno que contiene las funciones propias del servidor, uno que contiene las funciones relevantes del juego de tateti implementado y otro con las funciones auxiliares para el parseo de cadenas. Adicionalmente se adjunta un cliente en erlang para el testeo del servidor
La información que almacena el servidor se encuentra distribuida de la siguiente forma:

- *dispatcher* se encarga de iniciar los procesos en el nodo. Si se llama con el átomo *servidor* entonces dicho nodo se comportará como un servidor y almacenará usuarios y partidas. En cambio, si se llama con el átomo *trabajador* solamente se encargará se de ejecutar distintos *pcomando*

- *psocket* se encarga de escuchar en un socket por comandos que envíe el cliente, y según el identificador que recibió hará un spawn de *pcomando* con los argumentos correspondientes en el nodo indicado por *pbalance*. Al comienzo se llama con el socket donde deberá escuchar por mensajes y un átomo *noname* para indicar que solamente puede aceptar el comando CON o BYE. Cuando un usuario se registre, pasará a llamarse otra vez con el socket y de segundo argumento el nombre de usuario del jugador.

- *pbalance* almacena un mapa con la carga de los nodos y puede devolver el nodo más óptimo si se lo pide

- *pcomando* recibe argumentos como una tupla con la siguiente estructura: $\{atomo, [argumentosdelcomando], usuario, psocket\}$ donde *atomo* es un átomo que identifica el comando a realizar, seguido de una lista con los argumentos del comando, *usuario* es el nombre de usuario que almacena *psocket*, y *psocket* es el PID del proceso llamante. Cada *pcomando* realiza una acción distinta según el átomo que recibe

- *usuarios* es un proceso que se encarga de mantener un mapa de usuarios donde la clave es el nombre del usuario y el valor es una tupla que contiene el PID del psocket que se encarga de escuchar los pedidos del usuario y una

lista de los juegos en los que está participando actualmente. El nombre del usuario tiene un formato particular:NombreDelUsuarioNombreDelNodo@NombreDeLaPc. Supongamos que el usuario decidió llamarse "Seldon", el nodo donde se está almacenando su nombre se llama "Fundación" y la pc donde está Fundación se llama "Trántor", entonces su nombre de usuario será Seldon#Fundación@Trántor. De esta manera se pueden repetir los nombres de usuarios entre distintos servidores. Este proceso realiza operaciones para acceder y modificar dicho mapa.

- *juegos* es, al igual que *usuarios*, un proceso que se encarga de mantener un mapa con los juegos activos. Su clave será el id de un juego, y el valor una tupla $\{local, visitante, [espectadores], procesoDeJuego\}$ donde *local* es el id del jugador que inició el juego, *visitante* es el id del jugador que aceptó el juego, $[espectadores]$ es una lista de los espectadores del juego y *procesoDeJuego* es un PID al proceso encargado de manejar el juego de tateti. EL id del juego también tiene una forma particular: ContadorNombreDelNodo@NombreDeLaPc. Así, el primero juego en crearse en el nodo Fundación sería 1#Fundación@Trántor

La función *juegos* tiene una versión por si es llamada por nodos, en la cuál no hace nada más que responder un átomo *empty* a los mensajes que les llegan.
En el módulo dedicado a la implementación del tateti, la función principal *tateti* recibe como argumentos: El tablero con el que se esta jugando, el número de jugadas realizadas, el jugador que debe jugar y el id del juego que se está jugando.

2 Comunicación

Entre el cliente y el servidor la comuniación se realiza mediante strings correctamente formateadas sobre un socket TCP como lo indica la cátedra. Las jugadas se identifican como un entero entre 1 y 9, y el tablero es una lista compuesta de 1,0 o -1 donde 1 identifica una jugada del jugador local, -1 una jugada del jugador visitante y 0 una casilla disponible para hacer una jugada.
Los comandos exitosos se comunican a *psocket* como una tupla $\{ok, string\}$ donde *ok* indica que el comando fue exitoso y *string* es una cadena de la forma "OK cmdid op1 op2...". En cambio, si el comando no fue exitoso se enviará una tupla $\{error, string\}$ donde *error* indica que el comando no fue exitoso y *string* es una cadena de la forma "ERROR cmdid op1 op2..." donde op1 es la razón del error. *string* será finalmente enviada al cliente por el Socket.
Los errores posibles son: Usuario no registrado, usuario ya registrado, comando no existente, argumentos erróneos, formato erróneo, nombre ya registrado, juego ocupado, juego no existente, turno equivocado, jugada inválida.
La comunicación entre procesos que integran el servidor se da a través de llamadas del tipo $\{regName, Nodo\}$! *pedido*, donde generalmente obtenemos el nodo del nombre de usuario o desde el id del juego utilizando la función

obtieneNodo que recibe una cadena del tipo $algo\#Nodo$ y devuelve $\{ok, Nodo\}$ si fué correcto y el nodo existe, de lo contrario devuelve $\{error, Razón\}$.

3 Especificaciones

El nombre de usuarioo ingresado por el cliente no puede contener los siguientes carácteres:

- '@' (Arroba)
- '#' (Hash)
- '/' (ForwardSlash)
- ',' (Coma)
- ' ' (Espacio)

La ejecución de los archivos debe darse de la siguiente manera:

1. Suponemos que los nodos erlang se encuentran conectados entre sí
2. Si se quiere levantar un nodo servidor, se debe ejecutar $servidor : dispatcher(servidor)$. En cambio, si se quiere comenzar un nodo para aliviar la carga, se debe ejecutar $servidor : dispatcher(trabajador)$
3. Para conectar un cliente, se debe ejecutar $cliente : main(IP, Puerto)$ donde IP debe ser una string de la forma "N.N.N.N" o una tupla {N,N,N,N}, y Puerto debe ser un puerto donde se encuentre escuchando un servidor.