Trabajo Práctico Final

Sistemas Operativos I

Dzikiewicz, Luis Enrique
Legajo: D-3850/4
luisdzi.87@gmail.com

Ernandorena, Iván
Legajo: E-1115/1
ivan.ernandorena@gmail.com

Güella, Julio
Legajo: G-5061/1
julioguella@hotmail.com

Fecha de entrega: -2017

Docentes

Guido Macchi
Guillermo Grinblat
José Luis Díaz

Diseño del trabajo práctico

El trabajo práctico consta de tres archivos que permiten su funcionamiento:

- `server.erl`: En él se encuentra todo lo relacionado a los nodos servidores. Se tratan las conexiones de los usuarios, creación de juegos, jugadas, y demás comandos.
- `client.erl`: Aqui se encuentra lo necesario para que un usuario se conecte y comunique con el sistema distribuido. Llegan mensajes cortos y significativos que envia el servidor y los transforma en cadenas mas largas para la interpretación del usuario.
- `game.erl`: Se encarga de la interfaz y de la parte visual relacionada a el juego de TA-TE-TI, por ejemplo, se encuentran las funciones encargadas de imprimir la ayuda, el tablero, chequear cuando un jugador gana, etc.

Conceptos de diseño

- Al inicializar un servidor, se crea un nuevo proceso donde se llevará una lista con los nombres de los clientes de todo el sistema distribuido llamado `list_of_client`, que se registra globalmente con el nombre de `clients_pid` para poder acceder a este proceso desde cualquier nodo. Se realiza lo mismo, para un proceso registrado globalmente como `games_pid` para llevar los nombres de las partidas en curso. Además tanto los clientes como las partidas en curso han sido registradas globalmente. Se da inicio y registro local a los procesos de `pbalance` y `pstat`. Se inicia y registra globalmente el proceso `cargas` que sera encargado de mantener un diccionario actualizado, gracias a `pstat`, de los nodos del sistema con su respectiva carga. Luego de iniciar estos procesos claves para el almacenamiento de información y funcionamiento del sistema se pasa a la función `dispatcher` con el Socket donde el nodo recibe conexiones entrantes.(1)
- Con respecto al balanceo, la función `pstat` se encarga de enviarle a la función `cargas` el estado de carga del nodo (recordar que `pstat` esta registrada localmente). Luego, cuando un proceso le solicita a `pbalance` el nodo de menor carga, esté se lo solicita a `cargas` y luego se lo envia al proceso que lo solicito.(2)
- Un juego consta de sus jugadores, observadores, el tablero de juego, y de quien es el turno actual. Los jugadores dentro de un juego estan almacenados de la forma {`N,Jugador`}, donde `N` es un 1 o un 2 correspondiente al turno y `Jugador` es el nombre registrado globalmente (un átomo). Por diseño, el usuario que crea el juego con el comando `NEW` va a ser el jugador número 1 y siempre comienza a jugar el usuario que creo el juego, es decir jugador número 1.
- Cuando se realiza un cambio en un juego, el servidor envía automáticamente a cada jugador y observador de ese juego el nuevo tablero indicando quien fue el jugador que realizó esa jugada. Se obvió el comando `UPD` por esto, ya que esta alternativa parece más clara y eficiente para los usuarios.

Inicializar el sistema

- Antes de inicializar la consola de Erlang, ejecutamos el comando `$epmd -daemon` por si el comando `$erl` arranca automaticamente el epmd (Erlang Port Mapper Daemonepmd). Se hace para evitar el error "register/listen error: econnrefused".
- Antes de abrir la terminal de Erlang con `$erl` debemos conocer la IP del equipo en la red. Una de las formas de obtenerla es con el comando `$ipconfig`. Una vez que tenemos la IP abrimos la terminal de Erlang de la siguiente manera: `erl -name nodo@IP`, donde `nodo` es el nombre que le queremos dar al mismo y `IP` la que se obtuvo anteriormente.
- Compilamos el servidor con `c(server).` El servidor se arranca con la función `start\3` del módulo `server`. Esta función toma como parametros un nombre para el nodo (puede ser el mismo con el que se abrio el shell), un puerto disponible y una lista con el nombre de los demás nodos de la forma `'nodo@IP'` que en principio puede ser vacia ya que no hay otros nodos en el sistema.

- El sistema deberia sincronizarse automaticamente con los demas nodos de la red (si los hubiera), para chequear esto se puede utilizar la funcion `nodes()` que nos devuelve una lista de todos los demas nodos del sistema.

Como jugar

- Un usuario para conectarse deberá poseer el archivo `client.erl` y conocer la IP de uno de los nodos del sistema y su puerto. Una vez que se conoce esto, compilamos el archivo en una terminal de Erlang con `c(client).` y nos conectamos con la función `client:con\2` pasandole como argumento la IP y el puerto del nodo.
- Lo primero que se debe hacer es registrarse con un nombre de usuario utilizando el comando (`CON [nombre]`). Luego de esto se puede comenzar viendo los juegos disponibles con `LSG` o creando un nuevo juego `NEW [nombre juego]`. Para ver todos los comandos disponibles una vez conectado está disponible la ayuda con `HELP`.
- Al estar permitido participar en más de un juego a la vez. Cada vez que se muestra un tablero de un juego, en la parte superior se muestra el nombre del juego y quienes son sus jugadores junto con que simbolo tiene asignado cada uno. Además se indica quien fue el último en realizar una jugada, sobre todo para que los observadores puedan seguir mejor el progreso de una partida y si un jugador está jugando varias partidas simultaneamente pueda saber con exactitud a que juego pertenece el tablero que se le muestra.

Posibles expansiones

- Una funcionalidad que se podria agregar al sistema es que los jugadores pueden comunicarse entre ellos durante un juego en curso, asi como tambien los observadores del mismo puedan hacer comentarios sobre la partida.
- Una vez un usuario se conecto al sistema podria ver que otros usuarios estan conectados en ese momento, ademas de los juegos disponibles. Teniendo la posibilidad de enviarle un mensaje especifico a otro usuario. Generalmente conocido como `whisper` en juegos de tipo MOBA o MMORPG.
- Cuando un nodo del sistema se cae o se desconecta todos los clientes que tenian su `psocket` en ese nodo son desconectados del sistema. Se podria implementar que cuando un nodo se caiga los clientes que estaban siendo atendidos por este, sigan conectados al sistema, de forma transparente para el cliente.
- Tambien se podria implementar que cuando comienza un juego, no sea siempre el jugador que creo la partida el que comience, sino que sea aleatorio entre ambos jugadores.

---

(1) La bandera en `server` se utiliza para que solo se de registro global una sola vez en el primer nodo. A los demas nodos se les notifica de la existencia de estos procesos automaticamente.

(2) Eventualmente `cargas` podria ser el que directamente envie el nodo de menor carga, se mantiene asi para mantener la estructura pedida. No se calcula el minimo en `pbalance` para no tener que enviarle todo el diccionario.