FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

SISTEMAS OPERATIVOS I

TRABAJO PRÁCTICO FINAL

Andrés Imlauer

Introducción
En este informe se explicará a grandes rasgos la arquitectura del servidor y del cliente de
Ta-Te-Ti, además se explicarán como compilar, conectar nodos e iniciar el servidor, para finalizar las futuras posibles mejoras al servidor.
El servidor y el cliente consisten de los siguientes archivos: client.erl, dispatcher.erl, game_functions.erl, pbalance.erl, pcomando.erl, psocket.erl, pstat.erl, server.erl.

Arquitectura del servidor
La función que comienza a realizar la interacción con el cliente es **dispatcher:start**, que espera nuevas conexiones, luego crea un nuevo proceso llamado loop, donde se creará un nuevo hilo **psocket** por cada consulta del cliente, a su vez **psocket** por cada pedido del cliente crea un nuevo proceso **pcomando**, éste para no sobrecargar ningún nodo utiliza una funcion **nodo_libre**, que retorna el nodo con menos carga en el momento que el cliente hace la petición, luego **psocket** inicia **pcomando** en el nodo anteriormente seleccionado. Finalmente la función **pcomando** ejecuta el comando enviado por el cliente.
Cada nodo posee dos procesos, **pstat** se encarga de mandar intervalos regulares la información de carga del nodo al resto, **pbalance**, recibe esta información y calcula cual nodo debe recibir los siguientes comandos.

Implementación de las funciones encargadas de realizar una nueva partida
Cuando un usuario hace la peticion para crear una nueva partida usando el comando *new* lo que se hace es agregar la partida a una lista de juegos disponibles en el servidor, luego llama a la funcion **administrador**, lo que hace esta funcion es llamar a una nueva funcion llamada **nueva_partida**, cuando me encuentro en **nueva_partida** espero que el oponente me mande el comando *acc* (por lo tanto si no se acepta la jugada no se puede jugar aunque la partida este en la lista de juegos disponibles), luego de que el usuario acepte la jugada se creará una nueva función llamada **espero_guardo**, la cual mantiene una lista local por cada partida creada en distintos nodos.

Implementación del cliente
En lo que respecta al cliente, es bastante sencillo, su funcionamiento sólo consiste en identificar los comandos del usuario, a estos agregarle un número de identificación, enviarlos y esperar las respuestas del servidor con la función **receive_data**.

Compilando e iniciando el servidor y el cliente
En este ejemplo, iniciaremos un servidor local con dos nodos:
Primero, para iniciar el intérprete de Erlang escribimos erl y compilaremos el servidor escribiendo: c(servidor).
Luego abrimos diferentes terminales en la misma ubicación donde escribiremos: erl -sname ast y en la otra terminal: erl -same linus, también se puede usar: erl -name ast@IP.
Para conectar a los dos nodos, escribimos: net_adm:ping('linus@tu_host'), nodes() para comprobar que los nodos esten conectados.
Ahora si todo fue bien, escribimos: servidor:dispatcher_start(número de Puerto) en cada terminal utilizando diferentes puertos, y el servidor ya debería comenzar a funcionar.
Ahora el cliente, lo compilamos con c(client) , y nos conectamos a cualquiera de los dos nodos con client:start("127.0.0.1",Nº de Puerto).
La documentación de como jugar está explicada en el enunciado del trabajo, con lo que se refiere a los comandos disponbiles puedes consultarlo enviando el comando HELP desde el cliente.

TODO
- Implementar una función para que elija todo el tiempo partidas disponibles automáticamente, cuando finalice se conecte a otra, de no existir ninguna partida, creala. Sin necesidad de interactuar con el usuario.
- Agregar una funcion para dar turnos de forma aleatoria.· Desarrollar un cliente que posea entorno gráfico
- Implementar un comando para ver perfiles de cada usuario, manteniendo información de por ejemplo la fecha de registro, país, partidas perdidas, ganadas, si abandona muchas partidas, etc.
- Agregar función para que el cliente se actualice automáticamente.
- Agregar tiempo límite.
- Chequear que el usuario esté conectado fuera de cada comando.
- Escribir el cliente usando escript