Sistemas Operativos I - LCC
Trabajo Práctico Erlang

El trabajo se podrá realizar en grupo de hasta **dos** estudiantes. De ser así se deberán poner en contacto con los docentes de la cátedra indicando cómo se forma el grupo. Se entregará digitalmente enviando una copia digital a los docentes, junto con un informe corto de a lo sumo 3 página explicando las soluciones y decisiones tomadas.

Fecha de Entrega: **22 de Junio**.

Servidor de juegos distribuido

Un servidor de juegos sirve para que varios participantes, a través de aplicaciones cliente, puedan participar de juegos multijugador. Pueden ser de juegos de mesa, donde típicamente dos clientes participan mientras que un número variable observa la partida. El servidor se encarga de mantener todas las partidas correspondientes al mismo tiempo.

Dependiendo de la popularidad del servicio, puede ser necesario atender a miles de conexiones concurrentes y mantenerlas al tanto de los cambios en el estado de los juegos. Por eso se necesita hacer un diseño escalable y distribuido.

En este trabajo se deberá hacer un servidor de partidas de Ta-te-ti. Este servidor le permitirá a los usuarios conectarse, anunciar nuevas partidas y aceptarlas, así como unirse a una partida iniciada o por iniciar como observador. El servicio estará dado por varias máquinas corriendo el mismo código que deberán ponerse de acuerdo para dar a los clientes la impresión de un servidor único. Pueden ser representadas por distintas máquinas virtuales de Erlang ejecutándose en la misma PC.

Protocolo de comunicación

Las máquinas deberán actuar como un servidor escuchando en el puerto TCP 8000 (en caso de usar una misma PC, pueden escuchar en diferentes puertos). La comunicación entre cliente y servidor será mediante el envío de strings sobre un socket TCP.

La forma general de los pedidos del usuario al servidor será:

```
CMD cmdid op1 op2 ...
```

donde `CMD` son tres letras que indican la operación a realizar seguida de un identificador y posiblemente de sus argumentos. La respuesta del servidor será de la forma:

```
OK cmdid op1 op2 ...
ERROR cmdid op1 op2 ...
```

donde `OK` indica que el pedido identificado por `cmdid` se realizó correctamente devolviendo el resultado si lo hubiera como argumento y `ERROR` indica que hubo algún error al ejecutar el pedido.

De la misma manera, los mensajes que el servidor deba mandarle al cliente tendrán la forma:

```
CMD cmdid op1 op2 ...
```

a lo que el cliente deberá responder con:

```
OK cmdid
```

Las operaciones que permite al usuario son:

- `CON nombre`. Inicia la comunicación con el servidor. El cliente debe proveer un nombre de usuario. Si el nombre no está en uso, el servidor debe responder aceptando el pedido.
- `LSG cmdid`. Lista los juegos disponibles. Estos son los que están en desarrollo y los que están esperando un contrincante.
- `NEW cmdid`. Crea un nuevo juego, que será entre el jugador que lo cree y el primero que acepte.
- `ACC cmdid juegoid`. Acepta el juego identificado por `juegoid`. El servidor deberá contestar si el comando fue exitoso o no (por ejemplo, si alguien aceptó antes).
- `PLA cmdid juegoid jugada`. Realiza una jugada en el juego identificado por `juegoid`. La jugada puede ser abandonar el juego. Puede devolver error si la jugada es ilegal. En caso de ser aceptada, el servidor debe contestar con el cambio en el estado del juego.
- `OBS cmdid juegoid`. Pide observar un juego. El servidor le empezará a mandar los cambios en el estado del juego. El servidor debe contestar con el estado actual del juego.
- `LEA cmdid juegoid`. Deja de observar un juego.
- `BYE`. Termina la conexión. Abandona todos los juegos en los que participe.

Los mensajes que manda el servidor son:

- `UPD cmdid juegoid cambio`. Cambio en un juego. Lo reciben los observadores y el jugador contrario.

Arquitectura

Cada nodo del sistema ejecutará el mismo código, aunque pueden recibir distintos parámetros en la inicialización, por ejemplo los nombres de los otros nodos del sistema y opcionalmente el puerto en el que deba escuchar.

Cada uno contará con un proceso que espere nuevas conexiones (*dispatcher* en las figuras). Cuando un cliente se conecta, este proceso creará un nuevo hilo (*psocket*) que atenderá todos los pedidos de ese cliente. Por cada pedido, *psocket* creará un nuevo proceso (*pcomando*) que realice todo cálculo necesario y le devuelva una respuesta a *psocket*, que le enviará al cliente. Además *pcomando* se encargará de generar los mensajes correspondientes para el resto de los clientes y mandarlos a sus respectivos *psocket*, de ser necesario.

Para balancear la carga entre los nodos, cada proceso *pcomando* se creará en el nodo que se considere menos cargado en ese momento. Para eso, cada nodo tendrá dos procesos, uno encargado de mandar a intervalos regulares la información de carga del nodo al resto de los nodos (*pstat*), y otro encargado de recibir esta información y de calcular con una política simple cuál es el nodo que debe recibir los siguientes comandos (*pbalance*). Este último también recibirá pedidos de cada *psocket* preguntando dónde se debe crear el siguiente *pcomando*.

Cada nodo debe poseer solo parte de la información acerca de partidas y jugadores.

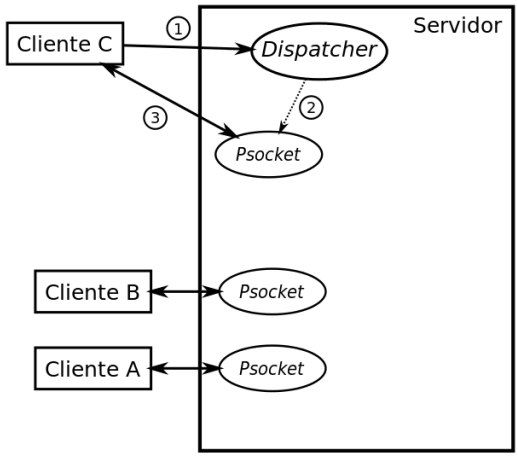

Figura 1: Conexión de un cliente. 1) El cliente se conecta al puerto que está escuchando *Dispatcher*. 2) *Dispatcher* crea un nuevo proceso para atender a ese cliente. 3) Todos los pedidos se hacen a través de la conexión TCP entre el cliente y el nuevo proceso. Las flechas contínuas indican conexiones TCP. Las punteadas, creación de un proceso.

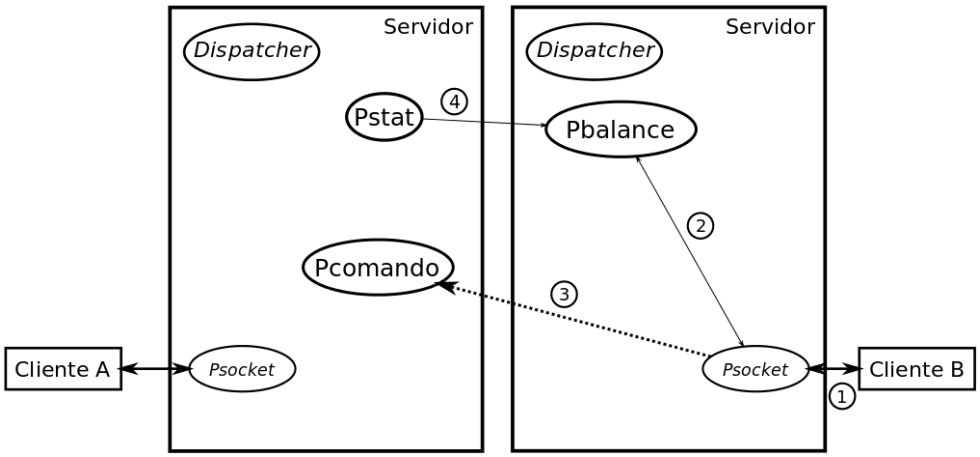

Figura 2: Procesamiento de un pedido. 1) El cliente hace un pedido a través de su conexión TCP. 2) El *psocket* encargado le pregunta al proceso *pbalance* de su nodo dónde se debe procesar el pedido. 3) *Psocket* crea el proceso que se encargará del pedido. 4) El proceso *pstat* de cada nodo le envía a itervalos regulares información de carga a los *pbalance* de los otros nodos. Las flechas finas indican intercambio de mensajes.

Objetivo

El estudiante deberá implementar el servidor mencionado. Para eso deberá:

- Definir la estructura de módulos y procesos. En particular, definir qué procesos serán los encargados de mantener la información del servidor.
- Analizar y seleccionar las estructuras de datos necesarias para representar tanto jugadores, juegos, etc.
- Implementar el manejo de la conexión TCP del servidor en el modulo dispatcher. Definir detalles del protocolo de comunicación con los clientes: cómo se pasan jugadas y estado, cuáles son los errores reportados, etc.
- Definir el protocolo de comunicación entre todos los procesos que integran el servidor.

Sugerencias

- Se puede empezar implementando el dispatcher, psocket y una versión de pcomando que siempre termine con el mensaje `ERROR no implementado`.
- Para implementar la conexión TCP en Erlang puede consultar el capítulo de sockets en [1].

- La función `erlang:statistics/1` [2] se puede usar para medir la carga de un nodo.

Referencias

[1] Francesco Cesarini y Simon Thompson, *Erlang Programming A Concurrent approach to software development.*

[2] Joe Armstrong, Robert Virding, Claes Wikström y Mike Williams, *Concurrent Programming in Erlang.*