Trabajo Práctico
Sistemas Operativos I

Santiago Iwakawa, Nicolás Del Piano

ERLANG

El programa del servidor se encuentra en el archivo *dispatcher.erl*. Allí se encuentra el proceso listener, la creación de los procesos socket para cada cliente, un contador para los descriptores y la inicialización de los workers. Es decir, todo lo que necesita el servidor para estar funcionando.

En el archivo *worker.erl*, se encuentra el código que describe cómo se estructuran los archivos, qué hace un worker y cómo lo hace y las funciones auxiliares.

Bien, empezamos por la estructura de nuestros archivos. Un archivo en nuestro servidor es una tupla {Fd,Nombre,Datos,P,B}, donde:
-Fd es un entero único para cada archivo cuando se crea (No cuando se abre, sino se le asigna cuando es creado) utilizando el proceso contador (función cont/1 en *dispatcher.erl*).
-Nombre es una lista de caracteres, sirve para identificar el archivo.
-Datos, una lista de caracteres donde se encuentra lo que escribimos en el archivo.
-P es una lista, que indica si el archivo está abierto por algún proceso socket o no. Si está vacía se puede abrir, si no, la lista contiene el proceso que la está usando.
-B, bytes leídos del archivo hasta el momento. Cuando se cierra se pone en 0.

Los workers se inician con el servidor, guardando sus respectivos pids en una lista de tamaño igual a la cantidad de workers. Cada uno de ellos tiene una lista de sockets (clientes conectados al mismo), y una lista con archivos. La comunicación entre los workers es directa mediantes mensajes de Erlang. Tenemos el conjunto de pids de los workers en una lista, y para acceder a otro worker simplemente proyectamos algún elemento de esa lista, eliminando de ésta el pid del worker emisor. Así, pasamos una única vez por los diferentes workers mandando mensajes de Erlang, realizando las operación enviada inicialmente por el cliente.

El trabajo de un worker, consiste en parsear el paquete enviado por el cliente a través del proceso socket, en listas de strings. Luego tomamos esa información parseada para interpretarla y hacer lo que el cliente pida.

El servidor de Erlang se inicia con la función *dispatcher:init()*.

POSIX Threads

En este caso los workers se comunican mediante el puerto *TCP*, básicamente el servidor opera con el siguiente protocolo:

-Los workers piden conectarse al servidor, enviando un mensaje indicando que son workers, y reciben a modo de confirmación de conexión su *ID* dentro del servidor.
-El dispatcher no permite que se conecte un cliente hasta que todos los workers estén listos.
-Cualquier conexión después de que los workers estén listos es considerada un cliente.
-Dentro del servidor hay un proceso por cada conexión.
-Todos los mensajes pasan por el servidor, los emitidos por los clientes, son enviados a los correspondientes workers.
-A los mensajes originales se le agregan argumentos, indicando tipo de receptor (cliente o worker), *ID* del receptor, y en caso que se necesite una respuesta colectiva de los workers, se indica número de workers que leyeron el mensaje, también se le agrega al mensaje la respuesta de cada worker, cuando las respuestas necesarias son recolectadas se le responde al cliente.
-La implementación del *FS*, esta hecha sobre archivos del *FS local*.

Algunas consideraciones: si un cliente manda *"BYE"* y no cerró sus archivos, el servidor se encargará de esta tarea. Al igual que si termina abruptamente.

En la parte de *POSIX*, se podrían haber asignado dinámicamente la memoria para los strings, pero en este caso estamos limitados por mensajes de *200* bytes.

Para compilar la parte de *POSIX*, hacemos un *make* en la terminal, y después iniciamos el servidor con el comando *sh FS.sh* en la terminal. Ahí se iniciarán el servidor, los workers, y finalmente un cliente. Podemos conectar los clientes que deseamos, en una nueva terminal, ejecutando *./Cliente*.