Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Trabajo Práctico Final

Sistemas Operativos 1

SISTEMA DE ARCHIVOS DISTRIBUIDO

Gabriel Antelo - David Giordana - Maximiliano Ibalborde

Este trabajo práctico tiene como objetivo implementar un sistema de archivos distribuido (FS), que permite , a priori, almacenar y compartir archivos de texto plano en una red de computadoras.
El Sistema de archivos actúa como servidor. Este escucha el puerto TCP 8000 a donde los usuarios (clientes) deberán conectarse. La comunicación se basa en el envío y recepción de strings sobre un socket TCP. El servidor por defecto soporta un conjunto de comandos básicos que permiten a los usuarios crear, manipular y eliminar archivos.
Existen dos implementaciones independientes, una desarrollada en Erlang y otra hecha en C y utilizando la librería POSIX pthreads.

Implementación en Erlang

Estructura General

Cuando el servidor está inicializado y listo para operar contamos con seis hilos en funcionamiento. Los primeros cinco se encargarán de administrar los archivos del sistema y responder a las solicitudes de los clientes. Por otra parte el hilo restante es el que se encargará de recibir las conexiones de los nuevos clientes. A este último lo conoceremos como dispatcher. Al conectarse un usuario el dispatcher creará un nuevo hilo para que atienda a sus mensajes y haga de intermediario entre el usuario y el worker que se le será asignado de manera aleatoria.
Cada cliente tendrá asignado un único worker. Este deberá resolver todas las solicitudes que reciba e interactuará con los demás workers dando así una falsa sensación al cliente de que está tratando con una sola unidad.
Los clientes al conectarse deben confirmar que quieren conectarse.Para ello el cliente tiene que ingresar el comando CON. Estando ya confirmado el cliente puede realizar las operaciones con los archivos. Entre los comando básicos tenemos:

- LSD
- RM <Filename>
- CRE <Filename>
- OPN <Filename>
- WRT FD <FileId>
- SIZE <Int> <Buffer>
- REA FD <FileID> SIZE <Int>
- CLO FD <FileID>
- BYE

¿Como funciona sistema de archivos?

Al lanzarse el programa se llama a la función filesystem.inicio. Esta se encargará de crear los 5 workers y llamar a la función filesystem.server(). A esta altura tenemos dos posibilidades, poder lanzar el servidor o que ocurra un error. En caso de ser lo segundo la causa principal sería que el puerto está en uso por lo que se imprimirá un mensaje de error y terminará la ejecución.
Si todo continúa sin invenientes se lanza el dispatcher (filesystem.dispatcher()) que se quedará esperando a una nueva conexión. Cuando un cliente se conecte llamará a un atendedor de clientes (filesystem.cliente()) y se lanzará un nuevo dispatcher.
En primera instancia dentro del atendedor de clientes se esperará un mensaje de conexión (CON). Cuando se lo reciba se le asignará un worker al azar y se procederá a ejecutar el cliente completo. Si el mensaje recibido es cualquier otro al que se esperaba se imprimirá un mensaje de error y se procederá a esperar nuevamente un mensaje.

El ciclo de vida de un cliente se basa en recibir un mensaje desde la red, parsearlo y enviarlo a un worker. Luego de esto quedará bloqueado esperando a una respuesta la cual devolverá mediante el socket utilizando un String.
El worker, por su parte, podemos dividirlo en dos bloques. El primero se encargará de realizar las acciones internas del worker, es decir, realizar el procesamiento que este pueda. En caso de ser necesario enviará solicitudes a otros workers. El segundo tiene como función procesar mensajes recibidos tanto de clientes como de workers y responder a los mismos.

Algunos detalles de los workers:

El worker funciona como si de un ciclo infinito se tratase. Para poder hacer correctamente el trabajo se almacena cierta información:

- PidList: Lista con el PID del resto de workers
- Files: Lista que contiene la representación de archivos del worker
- Cache: Información utilizada entre iteraciones.
- RequestList: Lista de solicitudes de clientes a responder
- LastFD: Último descriptor de archivo
- MsgCounter: Índice del último worker al que se le envió un mensaje. 0 representa al propio worker (no se envía información alguna)
- Safe: Se utiliza para evitar enviar mensajes repetidos a los workers. Es útil cuando responder una solicitud requiere varias iteraciones.
- ToDelete: Lista de archivos a borrar.

Las solicitudes de los clientes se almacenan en un arreglo.
Los mensajes de los workers pueden ser categorizados en:
Solicitud (*Req): Son mensajes que envia el worker a otros workers para solicitar cierta información.
Respuesta (*Res): Mensajes enviados desde otro worker para responder a una solicitud previa.
Mensaje de cliente (clientmsg): Son mensajes que envia el atendedor de cliente al worker con una solicitud.

Módulos

Para simplificar el trabajo con el sistema de archivo y su comprensión se ha modularizado el código en tres archivos:

- Filesystem.erl: Es el que se utilizará para cumplir el rol de servidor contando con los workers, dispatcher y los antededores de cliente.
- Parser.erl: Contiene los métodos necesarios para procesar un String y retornar una estructura con la información preprocesada. Esta será utilizada por los workers para trabajar.
- Filemodule.erl: En su interior están todos los métodos encargados de trabajar con los archivos.

Ejecución

Para facilitar el trabajo con el servidor se provee un archivo Makefile. Para utilizarlo es necesario escribir “make -f Makefile”, o simplemente make. La utilidad del archivo radica en compilar el sistema de archivos distribuido y ejecutarlo.

Extensiones opcionales

Entre la lista presentada en el enunciado se escogió:

- Implementación de RM de tal forma que, si un archivo está abierto, sea borrado cuando lo cierren.

Implementación en C

La implementación en C del sistema de archivos es prácticamente análoga a la de erlang.

Decisiones de diseño y diferencias con la versión de erlang

Para el envío de mensajes se emplearon las colas asíncronas provistas en POSIX mqueue.
Cada worker tiene asignado una cola donde se almacenarán los mensajes de los clientes.
En este caso contamos con dos tipos de mensajes:
Mensajes de worker: Son de uso interno para los workers
Mensajes de cliente: Aquí llegarán los mensajes enviados por los clientes.
Debido a que los mensajes de workers y de clientes llegarán a la misma cola se emplearon dos categorías de mensajes, una que se corresponde con un mensaje original y otro que es una copia. Así quedan 4 prioridades de mensaje:

1. PRIOREWORK: Mensaje reenviado de worker
2. PRIOWORKER: Mensaje de worker
3. PRIORECLIE: Mensaje reenviado de cliente.
4. PRIOCLIENT: Mensaje de cliente.

El funcionamiento de los workers se divide en inicialización y bucle de trabajo. Este útimo a su vez consta de diversas partes:

1. Bloqueo de worker en caso de no tener mensajes que leer ni solicitudes que responder
2. Si hay alguna solicitud por atender se ejecuta el bloque de procesamiento interno: allí el worker trabaja para resolverla.
3. En caso de no haber tenido que atender una solicitud de cliente se revisa la cola en busca de mensaje de clientes. Si lo encuentra lo parsea.
4. Bloque de mensajes de worker: Aquí se recibirán y procesarán los mensajes enviados al worker.