Trabajo Práctico Final
Sistema de Archivos Distribuido
Sistema Operativo I

Rossi Federico
R-3796/6
frossi.933@gmail.com

May 16, 2014

1 Objetivo

El objetivo principal de este trabajo práctico es desarrollar un sistema de archivos distribuido simplificado con un set de operaciones mínimas, que sirve para compartir archivos, impresoras y otros recursos como un almacenamiento persistente en una red de computadoras. Todo cliente que quiera entrar al sistema debe establecer una conexión al IP donde se está ejecutando el servidor a través del puerto 8000. Luego puede realizar todas las consultas u operaciones que desee a través de los comandos que nombraremos mas adelante

2 Operaciones

Las operaciones que permite al usuario son:

- CON

  Inicia la comunicación con el servidor y este le devuelve un ID de conexión.

- LSD

  Pide un listado de los archivos existentes en el FS y el servidor responde "OK" junto con la lista de archivos separada por espacios y terminada por el caracter nulo.

- OPN *name*

  Abre el archivo indicado por el primer argumento. El servidor devuelve OK FD n donde n es un numero entero, el cual describe dicho archivo y lo usaremos luego para realizar las demás operaciones.

- DEL *name*

  Borra el archivo *name* del FS. El servidor responde OK si el archivo existe y nadie lo tiene abierto.

- CRE *name*

  Crea un archivo *name* en el FS. El servidor responde OK si es que este archivo no existía previamente.

- WRT FD n SIZE m *buff*

  Pide la escritura del buffer *buff* de tamaño m en el descriptor n. El servidor responde OK.

- REA FD n SIZE m

  Lee m bytes del archivo dado por el descriptor n. El servidor responde OK *buff* cuando la lectura fue correcta indicando que leyó el buffer *buff*. Si se llega al final del archivo, se devuelve la cadena vacía.

- CLO FD n

  Cierra el archivo n. El servidor responde OK.

- BYE

  Cierra la conexión (cerrando todos los archivos abiertos).

En cada caso de que alguna operación falle, el servidor responde con un mensaje de error, el cual detalla la razón del mismo.

3 Implementación

El sistema se encuentra implementado en dos lenguajes de programación bastante distintos entre si. Por un lado tenemos el sistema hecho en C, un lenguaje imperativo de nivel medio pero con muchas características de bajo nivel. Para esta implementación he utilizado la extension de POSIX para el manejo de múltiples hilos. Por otro lado, tenemos el sistema realizado en Erlang, un lenguaje de programación concurrente y funcional, hecho con el principal objetivo de realizar aplicaciones distribuidas, tolerantes a fallos, soft-real-time y de funcionamiento ininterrumpido. En cuanto a mi opinión personal, considero que en el segundo caso me he sentido mas cómodo y con muchas mas facilidades a la hora de desarrollar el trabajo, a pesar de tener muy poco experiencia con dicho lenguaje. Es verdad también, el lenguaje C nos brinda la posibilidad de implementar el sistema de manera mas eficiente en cuanto a velocidad o uso de recursos del sistema, pero Erlang nos propone un entorno mas cuidado en el que no debemos concentrarnos en detalles, de memoria por ejemplo, y nos permite enfocarnos mayormente en cuestiones importantes de este tipo de aplicaciones.

En ambos casos, para cada worker he utilizado dos tablas, la primera denominada "Tabla de Archivos" me indica los archivos presentes en el FS local y la segunda llamada "Tabla de Trabajo" contiene información de los archivos abiertos. Esta ultima posee los datos necesarios para distinguir entre archivos locales abiertos por clientes de dicho worker, archivos locales abiertos por otros workers y archivos externos abiertos por clientes de este worker. Esto permite que, a la hora de querer cerrar, leer o escribir un archivo externo, ya sepamos exactamente a que worker dirigirnos, en lugar de enviar la consulta por broadcast. Ademas, se simplifica la operación de apertura ya que solamente debemos fijarnos si el archivo local, que nos pidieron abrir, no se encuentra en dicha tabla sin la necesidad de realizar consultas a todos los demas workers o analizar localmente archivo por archivo.

A continuación hablaremos mas en detalle sobre las características particulares de cada implementación.

3.1 Lenguaje C

En este caso, el servidor cuenta con una serie de workers dispuestos en diferentes PCs y realiza las comunicaciones internas mediante los puertos TCP/IP. A continuación comentaré como es el proceso de inicio de todo el sistema. Primero la computadora central del servidor, a la cual se conectarán y enviaran los comandos los clientes, debe ejecutar el proceso *server*. Este proceso espera que se conecten los workers, para luego comenzar a aceptar las conexiones de los clientes y, paso siguiente, resolver sus consultas. Cada computadora que actúa como worker, debe ejecutar el proceso que lleva dicho nombre y pasarle como argumentos el IP y el puerto del servidor, en ese orden.

Una vez que se hayan conectado correctamente el número de workers dispuesto por el server, este le envía a cada uno de ellos la información necesaria para que se establezcan las comunicaciones entre los mismos. Estos mensajes, claramente, serán a través de los puertos TCP/IP y para esto he determinado un protocolo de comunicación para atender los pedidos de cada uno y obtener la respuesta deseada. Dicho protocolo consta de la estructura detallada en la Tabla 1.

Una de las cuestiones que he tenido que solucionar dentro del sistema es la sincronización de los workers, ya que la concurrencia dentro del sistema de archivos puede llegar a producir una falta de consistencia en los datos, una respuesta incorrecta al cliente, etc. Para esto, he decido utilizar un par de mutex para cada worker, que protegen el sistema de archivos local de cada uno de ellos. De esta manera, podemos tener varios hilos atendiendo las consultas de los clientes (o de otros workers) simultáneamente y solo debemos preocuparnos por que en cada instante a lo sumo haya un hilo accediendo al sistema de archivos, y esto justamente lo logramos con la exclusión mutua mencionada. También es importante que en todo momento solo un cliente pueda realizar la operación de creación de un archivo, por lo tanto, para evitar comportamientos indeseados dentro del sistema, he utilizado un lock centrado en el servidor. Es decir, cada vez que un worker tenga que realizar la creación de un nuevo archivo debe tomar posesión de dicho lock(para esto envía un mensaje "LOCK" al servidor) en caso de que nadie lo haya hecho ya, sino debe esperar a que sea liberado(esto se realiza enviando "UNLOCK"). Esta es la única operación en la cual debemos tomar este recaudo.

Table 1: *Protocolo de comunicación entre Workers*

Pedido	Respuesta	Descripción
WRK lsd	*Arch1 Arch2 ... ArchN*	Lista con todos los archivos del FS local del worker
WRK opn *ID NAME*	**ok** *FD*	Apertura correcta, *FD* es su descriptor de archivo
	e1	Error 1: el archivo no existe
	e2	Error 2: El archivo ya está abierto por otra persona
WRK clo *FD*	**ok**	Cerrado correctamente
WRK wrt *FD SIZE BUFF*	**ok**	Escritura correcta
	e3	Error 3: Falló durante la escritura
WRK rea *FD SIZE*	*BUFF*	*BUFF* es el resultado de la lectura
WRK del *NAME*	**ok**	Borrado correctamente.
	e1	Error 1: el archivo no existe
	e2	Error 2: El archivo está abierto
WRK exs *NAME*	**y**	Existe el archivo *NAME*
	n	NO existe el archivo *NAME*

3.2 Lenguaje Erlang

Para el caso de este lenguaje podemos aprovechar las características que mencionamos anteriormente para el desarrollo de la aplicación. Los workers no van a ser procesos corriendo en diferentes PC's, sino que son hilos creados por el proceso principal del servidor dentro de la maquina virtual de Erlang. Entonces, utilizo el sistema de envió de mensajes para comunicar los workers entre si y con el servidor. Esta es una gran herramienta que nos simplifica el trabajo a la hora de programar ya que no debemos concentrarnos en detalles de implementación. En cuanto a dicha comunicación, utiliza la misma estructura del protocolo detallado anteriormente.

Aprovechando otras facilidades que nos brinda Erlang, he decidido realizar el punto adicional que consiste en el reinicio del grupo de workers en caso de error o perdida de alguno de ellos. Aquí los pedidos que fueron realizados se pierden y el FS se reinicializa vacío. Para llevar a cabo este punto he utilizado la función de enlazar los procesos de forma informativa, para monitorizar el estado actual de cada uno y poder detectar si alguno de ellos finaliza. De este modo, si se detecta que alguno de los workers ha terminado, todos los demás son finalizados intencionalmente y se vuelve a iniciar el proceso del servidor desde cero. En este procedimiento se pierden los datos almacenados y las operaciones que se estaban llevando a cabo, pero logramos que el servidor se recupere de fallos dentro de los workers y que continúe funcionando. Como ejercicio para el futuro podría ser, implementar algún proceso de recuperación mas completo que resguarde los datos, para esto podríamos hacer que los datos se guarden de manera permanente en el disco por ejemplo.

Otra diferencia que se puede marcar, dentro de como funciona internamente el sistema, con respecto al desarrollado en C, es que tenemos un único hilo (por worker) que se ocupa de hacer lo esencial de cada operación y, ademas, de atender las consultas de los demás workers. En cambio, en el otro sistema lanzamos un hilo por cada cliente (por worker) que se encarga de las operaciones que este le solicita, y por otro lado, tenemos un hilo dedicado a responder consultas de los demás workers. Como se puede observar, en este ultimo caso, es necesario tener mucho mas cuidado en cuanto a la concurrencia y sincronización de los threads, pero con una implementación correcta obtenemos un mejor desempeño.