Sistemas Operativos I

Trabajo final

Baruffaldi, Juan Manuel
Pellejero, Nicolás
Perezzini, Luciano

Funcionamiento

Nuestro servidor, tiene la capacidad de que, múltiples clientes, puedan abrir un mismo archivo y acceder al mismo en forma de lectura y escritura; aparte de todas las funcionalidades pedidas por el enunciado del proyecto. Estas son CON, LSD, DEL, CRE, OPN, WRT, REA, CLO y BYE.

Además, en Erlang, implementamos la mensajería tolerante a fallas: si el FS no responde después de 100 ms, el hilo que atiende el pedido devolverá *ERROR numError ETIME.*

En ambos lenguajes decidimos iniciar la conexión sólo después de que el cliente solicitara el comando *CON*, detalle que no estaba contemplado en el cliente dado por la cátedra. Por esto, decidimos entonces dejar el servidor funcionando con *telnet*.

Forma de trabajo:

Para realizar el proyecto en C, utilizamos Posix Messages Queues. Creamos y abrimos una cola de mensajes para cada uno de los workers, es decir, se poseen 5 colas de mensajes.

En Erlang utilizamos los mensajes que provee el lenguaje.

Lo primero que se hace es levantar todos los procesos necesarios desde un principio.

En Erlang levantamos:

- Los 5 workers en anillo, para poder comunicarse entre sí.
- Los procesos cnt y cnt_error que son los contadores para los descriptores de archivos y los numeros de error.

En C levantamos:

- Los 5 workers en forma independiente, ya que en este caso se comunican entre ellos estilo broadcast.

Una vez que se conecta un nuevo cliente, le asignamos un nuevo ID, luego le asignamos un worker aleatorio, alguno de los 5. Este es el caso en donde aparece por pantalla OK ID X, donde X es el número del worker asignado. En ambos casos se acepta la conexión en un nuevo proceso.

Ahora es el momento de que el worker comience a recibir información del cliente. Es decir, se analiza por casos lo pedido por el cliente y se ejecuta el código correspondiente.

En C, le asignamos a cada comando una prioridad, y distinguimos también los mensajes entre workers, de los que son worker ↔ cliente.

Los archivos se guardan en memoria, es decir, tenemos una estructura cuyos atributos son el nombre del archivo y el contenido del archivo. Luego, hacemos algo parecido para manejar a los archivos abiertos: tenemos una estructura cuyos atributos son el ID del cliente, el nombre del archivo en cuestión, el ID del worker y la posición del cursor en el contenido del archivo.

En Erlang tenemos una lista de tuplas que representa el buffer de cada worker y otra para los archivos abiertos. Se incluyen una serie de datos necesarios para el funcionamiento del servidor, similares a los datos usados en las estructuras de C. Es decir, guardamos los archivos en memoria de la siguiente forma: *[{nombreTexto1, contenidoTexto1}, …]*. A los abiertos los guardamos en Abiertos.

También, en Erlang, tenemos una función *flush* para "limpiar" los *registers.*

<u>Análisis de los comandos:</u>

- LSD: Mandamos un mensaje diciendo que se pide LSD a todos los demás workers (menos al actual) con su ID de worker correspondiente. Cuando llega todo el contenido de los workers, concatenamos todos los archivos y lo mostramos en la pantalla del cliente.

- DEL: Nos fijamos que el archivo que se quiere eliminar no se encuentre entre los archivos abiertos. En caso de que esto ocurra, el cliente responderá EOPNFILE. Sino, pedimos la función DEL a los otros workers. Si el archivo no está abierto en nungún worker, procedemos a eliminarlo.

- CRE: Primero y principal, nos fijamos si el archivo ya está creado. Esto lo hacemos fijándonos si el nombre ya se encuentra en el buffer de los archivos. En caso de que esto suceda, devuelve EBADARG. Sino, mandamos un mensaje, a los demás workers, de que queremos ejecutar CRE. Incrementamos la cantidad de archivos en el buffer y en éste último, alojamos el nuevo archivo con contenido nulo.

- OPN: Al array Abiertos, en la posición NFD (la cual es 0 al comienzo del programa), le colocamos el nombre del archivo, el ID del worker, el ID del cliente y el cursor en 0. Luego vamos incrementando el NFD.

- WRT: Primero nos fijamos si el NFD dado como argumento está abierto. En caso de que no, devuelve EBADARG. En caso de que sí, procede y busca el nombre del archivo en el buffer de todos los archivos. Concatena el contenido viejo del archivo con el contenido dado con el comando.

- REA:Nos fijamos si el archivo está abierto. En caso de que no, devuelve EBADARG. En caso de que sí, manda el pedido a todos los demás workers. Funciona de la siguiente manera: supongamos que queremos abrir el archivo *Archivo1.txt* que se encuentra en memoria cuyo contenido es *texto*. Desde un cliente ejecuto *OPN Archivo1.txt*, lo cual devuelve *OK FD 0*. Ahora, ejecuto *REA FD 0 SIZE 1*, el programa va devolver *OK SIZE 1 t* e incrementa la posición del cursor en una unidad. Es decir, ahora el cursor se encuentra en la letra *e*. Si ahora ejecuto *REA FD 0 SIZE 2*, va a devolver *OK SIZE 2 et*c. En el caso de ejecutar *REA FD 0 SIZE 66*, el programa va a devolver *OK*; ya que 66 es más grande que la longitud del contenido que se encuentra o quedó en el *Archivo1.txt*.

- CLO:Nos fijamos si el archivo en cuestión está abierto; ya que no tiene sentido de que no lo estuviera. En el caso de éxito, “borramos” lo que hay en la posición del NFD del array Abiertos.

- BYE: se cierra la conexión con el cliente. Se cierra el socket correspondiente del cliente.

<u>Representamos los errores de la siguiente forma:</u>

- EBADFD: “el NFD dado no existe”
- EOPNFILE: “no se puede eliminar el archivo ya que está abierto por algún cliente”
- EBADARG: “no se conoce el o los argumentos del comando introducido”
- EBADCMD: “no se conoce el comando introducido”

La cantidad de errores ocurridos se van sumando, por ejemplo, si tenemos *ERROR 19 EBADFD*, quiere decir que hasta ese momento ocurrieron 19 errores.

Forma de ejecución en C:

Nuestro servidor escucha un puerto y se comunica mediante *telnet*. Con lo cual, supongamos que el archivo ejecutable se llame *server*, entonces en una terminal pondríamos lo siguiente: *./server 8000*, donde 8000 es un puerto. Ahora, para conectar un cliente haríamos lo siguiente: abrir otra terminal, y escribir *telnet localhost 8000.* De esta forma, podríamos conectar múltiples clientes.

Forma de ejecución en Erlang:

El código se llama *serverErlang.erl*, para ejecutarlo se deben seguir los siguientes pasos: en una terminal, abrir Erlang y compilar el código; ejecutar *serverErlang:listen(8000)* (donde 8000 es un puerto). Luego, abrir con *telnet* otra terminal de la misma forma que en Posix.