República Bolivariana de Venezuela
Universidad Simón Bolívar

Redes de Computadoras I

Informe Taller 3 Redes I

Autores:

Di Battista, Emma 13-10372

Navarro, Alejandro 13-10969

Perez, Gabriel 13-11104

Caracas, Octubre de 2017

Más allá del enfoque que se le quizo dar al proyecto al simular un sistema de monitoreo de transacciones sobre un cajero, el proyecto basicamente consistía en programar un servidor y un cliente TCP usando la API de sockets de Berkeley y la librería de hilos para tener cierto nivel de concurrencia.

La entrega consite ademas del informe de 3 archivos. Dos archivos con el código fuente del cliente y el servidor y un Makefile que los compila adecuamente con solo teclear make en la terminal.

El código fuente del cliente consiste en la implementación de un socket que se crea, conecta y envia mensajes al servidor, todo esto con su respectiva verificación de funcionamiento. En el caso de que falle alguna de las operaciones anteriores, se le anuncia al usuario y se cierra el programa del cliente.

Dado que así lo solicitan los profesores, al cliente se le debe indicar la direccion IP del servidor y el puerto por donde recibira la información. Hay un tercer parámetro que es totalmente opcional que el el puerto por donde se comunica el cliente. La sintaxis para correr el programa cliente seria:

svr_c -d <nombre_módulo_central> -p <puerto_svr_s> [-l <puerto_local>]

Puede ser tecleada en cualquier orden y también se puede no incluir el puerto local ya que es un parámetro opcional.

Por otra parte el código del servidor incluye muchas más funcionalidades y por lo tanto mas lineas de código.

Para empezar se define una estructura que representara a cada hilo y contiene cada uno de los parámetros que estos necesitan para llevar a cabo el proyecto. Entre estos se encuentran, el file descriptor que representará el socket por donde se comunicará cada hilo con un cliente, una cadena de caracteres que almacenará cada mensaje enviado por los clientes, una cadena de caracteres que almacena el nombre del archivo de salida donde se irán imprimiendo las bitácoras y por último un mutex que le permitirá a cada hilo ejecutarse

de manera atómica en las regiones críticas del programa, que en este caso sería cada vez que se debe escribir sobre el archivo de salida.

En cuanto a las funciones, se podria generalizar el funcionamiento tanto del main como de las otras dos de la siguiente manera:

En el main se verifica que se cumpla con la sintaxis deseada:

svr_s -l <puerto_svr_s> -b <archivo_bitácora>

y ultilizando la informacion proporcionada por el usuario se procede a crear el socket TCP escuchando por el puerto indicado por el usuario, recibiendo información de cualquier dirección IP.

Posterior a la inicialización del socket y su oportuna verificación de que se haya creado adecuadamente, el programa entra en un ciclo infinito donde por cada cliente que solicite conexión, y esta se acepte y establezca correctamente, se creará un hilo que será enviado a correr la función connection_handler.

En la función connection_handler, el hilo luego de utilizar la función setsockopt() para mantener una verificación activa del estado de la conexion, procede a entrar en un ciclo infinito donde ira recibiendo mensajes del cliente, pasandolos a una función que verifica si el mensaje contiene algún patrón que podría representar una alerta, y luego esta información es impresa en pantalla y en el archivo de salida de manera atómica gracias al uso correcto del mutex de cada hilo.

Por ultimo, cabe destacar que al no poseer un patron claro los mensajes de entrada proporcionados en el pdf con los requerimientos, se procedió a usar los siguientes sin alterar la sintaxis deseada de las bitácoras:

003:28:2012:01:38:200004 Communication Offline
003:28:2012:02:45:260004 Communication error
003:28:2012:03:52:420004 Low Cash alert
003:28:2012:04:39:250004 Running Out of notes in cassette
003:28:2012:05:41:000004 empty
003:28:2012:06:41:390004 Service mode entered
003:28:2012:07:38:200004 Service mode left
003:28:2012:08:45:260004 device did not answer as expected
003:28:2012:09:52:420004 The protocol was cancelled
003:28:2012:10:39:250004 Low Paper warning
003:28:2012:11:41:000004 Printer Error
003:28:2012:12:41:390004 Paper-out condition

Como se puede observar el cuarto campo numérico es el código de identificación de la bitácora, cada vez que una de estas lineas es transmitida del cliente al servidor, el mismo emitirá un mensaje de alerta tanto en la pantalla como en el archivo de salida. Cualquier otra transacción que no posea alguno de los códigos anteriores también sera impresa en pantalla y en el archivo pero sin mensaje de alerta.

De igual manera, se utilizó una función “timeout” que toma con argumentos el “filedescriptor” del socket y además, un entero que equivale a un valor de tiempo en segundos y; devuelve un entero que será 1 si el canal se mantiene activo; 0 si el canal se mantiene inactivo y; -1 si se encuentra un error. Esta función es utilizada para indicar al servidor que no ha llegado data proveniente del socket cuyo filedescriptor es el argumento de la función; cada vez que transcurra el tiempo establecido (5min en este caso) se solicita enviar mensajes para no ocupar espacio y procesamiento sin necesidad. Cabe destacar que la función solamente envía un aviso al servidor, mas no lo elimina de las conexiones activas, es decir, se cumple con uno de los requisitos de este taller. Para efectos de este taller, se configurará la espera de 10 segundos, para verificar el funcionamiento correcto y continuo de esta función.

Adicionalmente, cuando existe un mensaje de alerta emitido por el servidor, este se envía por correo electrónico al administrador, cuya dirección de correo debe ser

especificada en la variable de ambiente $MAILTO. Esto permite filtrar los mensajes, para que el administrador sólo reciba las alertas y trabaje para solucionarlas, sin tener que recibir mensajes innecesarios.

Si se desea un mayor detalle del funcionamiento del codigo del programa, se anexa a la entrega todo el código fuente detalladamente comentado.