Universidad Simón Bolívar
Departamento de Computación y TI
CI-3825 – Sistemas de Operación
Septiembre-diciembre 2015

Proyecto 1: *Chat*

Introducción

Un programa de chat permite la comunicación entre dos o más personas en tiempo real por medio de mensajes escritos. En este proyecto, se requiere que desarrolle un programa de chat entre usuarios de la misma computadora, basado en una arquitectura cliente/servidor. El cliente será el programa que interactúa directamente con los usuarios, y el servidor será el encargado de coordinar la comunicación entre los clientes.

En el sistema que usted debe desarrollar la comunicación será centralizada, en otras palabras, los clientes no se comunicarán directamente, sino que todos los mensajes pasarán por el servidor. La comunicación entre los clientes y el servidor se realizará por medio de pipes nominales.

En la sección que sigue se describe el programa cliente. Más adelante, se describirán el servidor y los detalles de la comunicación entre los clientes y el servidor.

Cliente

El cliente es el programa que interactúa directamente con el usuario y le permite mantener una o varias conversaciones. El área de la pantalla del cliente estará dividida en dos ventanas. La ventana de conversación, que estará en la parte superior, mostrará los mensajes enviados y recibidos por el usuario. La ventana de entrada, que estará en la parte inferior, será el área donde el usuario escribirá mensajes antes de enviarlos, así como órdenes del programa.

El programa cliente aceptará las siguientes órdenes. Los nombres de órdenes empiezan por un guión, con el fin de distinguirlos de las palabras de un mensaje.

-quien
-escribir <nombre de usuario>
-estoy <estado>
-salir

A continuación se explica cada orden por separado.

quien

Esta orden muestra una lista de los usuarios conectados al servidor y el estado de cada uno. El estado de un usuario es cualquier cadena de caracteres

elegida por el usuario para indicar si está presente o ausente, su estado de ánimo, etc.

La lista desplegada por la orden *quien* se mostrará en la ventana de conversación.

escribir <nombre de usuario>

Esta orden toma el nombre de un usuario conectado como argumento e indica que se quiere conversar con él. Los mensajes enviados después de ejecutar la orden serán dirigidos a ese usuario.

Si se vuelve a ejecutar la orden *escribir* con otro nombre de usuario, los mensajes ahora serán dirigidos al nuevo usuario. De este modo, será posible mantener varias conversaciones con distintos usuarios en la misma pantalla.

estoy <estado>

Esta orden cambia el estado del usuario.

Si un cliente está conversando con determinado usuario (es decir, si la última orden *escribir* ejecutada en el cliente fue a ese usuario), y este usuario cambia de estado, se debe mostrar una notificación en el cliente con el nuevo estado del usuario.

salir

Esta opción cierra el programa cliente y le notifica al servidor que el usuario se desconectó. Se mostrará un mensaje indicando que el usuario se desconectó en los clientes que estén conversando con él.

Formato de llamada

El programa cliente será llamado de la siguiente manera:

cliente [-p pipe] [usuario]

La opción `-p` es opcional e indica el nombre del pipe a través del cual el servidor espera por conexiones nuevas. Si no se especifica, se tomará por defecto como `/tmp/servidor`.

El parámetro *usuario* se refiere al nombre que tendrá el usuario en el chat. Si no se especifica, será el nombre de usuario del sistema.

Cuando un nuevo usuario se conecte al servidor, deberá aparecer en todos los clientes conectados una notificación de que el nuevo usuario se conectó.

Interfaz

En la página del taller se publicará el archivo fuente de un prototipo de la interfaz que deberá tener el cliente. Esta interfaz está basada en la biblioteca *ncurses*. En la documentación del archivo se proveen más detalles.

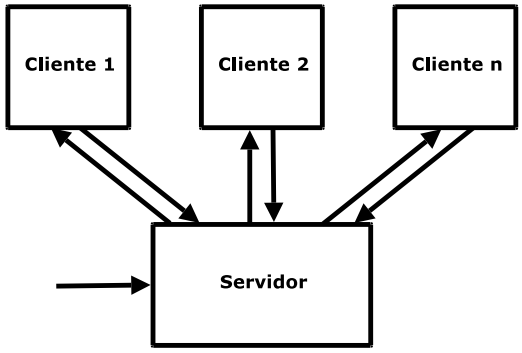

Figura 1: Topología de la red

Servidor

El servidor se encarga de dirigir la comunicación entre los clientes. Cuando un cliente envía un mensaje, el servidor lo recibe y lo dirige hacia el cliente de destino. El servidor deberá manejar los mensajes enviados por los usuarios, así como mensajes de control que indican el cambio de estado de un usuario, la conexión o desconexión de un usuario, entre otros.

En la figura 1 se ilustra la topología de la red. Cada cliente conectado se comunica con el servidor a través de dos pipes nominales, uno de entrada y el otro de salida. Además, el servidor tendrá un pipe nominal siempre abierto por el cual recibirá solicitudes de conexión nuevas.

Formato de llamada

El servidor será llamado de la siguiente manera:

servidor [pipe]

El programa toma como parámetro opcional el nombre del pipe a través del cual esperar por conexiones nuevas. Si no se especifica, será por defecto `/tmp/servidor`. Si trabaja de forma remota en una computadora del LDC, se aconseja que utilice un nombre único para usted, por ejemplo, `/tmp/servidorNumeroCarnet`.

Detalles de implementación

En esta sección se discuten algunos detalles técnicos sobre la implementación de los programas.

Ciclo de procesamiento de mensajes en el servidor

Como se discutió en la sección anterior, el servidor deberá atender los mensajes enviados por los clientes, los cuales llegarán al servidor en un orden no predecible. Por lo tanto, no es posible hacer *read* en secuencia de todos los pipes, porque la llamada *read* bloquea y el programa se quedaría indefinidamente esperando en la mayoría de los casos.

Una solución a este problema es la llamada al sistema *select*. Esta función espera simultáneamente por un conjunto de descriptores de archivo y devuelve cuando alguno de los descriptores de archivo cambia de estado, indicando cuáles son los descriptores que deben ser atendidos. Una explicación detallada de *select* se encuentra en 'man select_tut' (también en 'man select', pero la primera es más explicativa). Utilizando *select* es posible implementar el servidor como un programa monoproceso (sin utilizar varios hilos o procesos), lo cual simplifica notablemente la implementación.

Además de lo descrito anteriormente, la función *select* toma un parámetro adicional que indica un tiempo máximo por el cual la función debe bloquear. Por lo tanto, la función devuelve cuando alguno de los descriptores cambia de estado o bien cuando transcurre el tiempo máximo. Esta funcionalidad se utilizará en el servidor para revisar periódicamente si existen solicitudes de conexión nuevas.

El servidor estará basado en un ciclo principal que revisa cuáles descriptores han cambiado de estado y realiza las acciones correspondientes. La estructura del ciclo será la siguiente:

```
1. while (1) {
2.     Hacer select de los descriptores de entrada, con cota de tiempo 1 s.
3.     Si uno o varios descriptores de archivo cambiaron de estado:
4.         Realizar las acciones correspondientes.
5.     Revisar el pipe de conexiones entrantes.
6.     Si existe una solicitud de conexión:
7.         Establecer la comunicación con el nuevo cliente.
8. }
```

Establecimiento de la conexión entre cliente y servidor

El siguiente proceso permitirá que un cliente establezca la conexión con el servidor. Al iniciarse el servidor, este crea un pipe para recibir solicitudes de conexión. Este será un pipe nominal abierto en modo de lectura. Adicionalmente, el pipe debe ser abierto con la opción `O_NONBLOCK`, con la finalidad de que la llamada *open* no bloquee esperando que alguien lo abra en el otro extremo (al abrirlo con la opción `O_NONBLOCK`, la llamada *read* tampoco bloqueará).

Cuando un cliente desea conectarse con el servidor, lo primero que hace es crear dos pipes nominales (que serán los pipes de lectura y escritura) en una ubicación específica. A continuación, abre el pipe de conexiones entrantes del servidor y escribe en él la ubicación de los pipes recién creados. El servidor, al leer de este pipe, abre los dos pipes indicados. Por su parte, el cliente abre

también los pipes recién creados, y con esto queda establecida la comunicación.

Terminación inesperada del cliente

Si un cliente termina de forma abrupta (por ejemplo, como resultado de recibir la señal KILL), se cerrarán los extremos abiertos por el cliente de los dos pipes. Esto ocasionará que el pipe que estaba abierto por el servidor en modo de lectura devuelva repetidamente *end of file*. Por otra parte, si el servidor intenta escribir en el pipe de escritura, recibirá la señal SIGPIPE, cuya acción por defecto es terminar el proceso.

Estas dos situaciones se deben evitar haciendo que el pipe que repetidamente devolvería *end of file* se cierre, y que la señal SIGPIPE sea ignorada o capturada por un manejador.

Terminación del cliente por interrupción del teclado

En el prototipo de interfaz se puede apreciar que si el programa se cierra presionando Ctrl-C, el terminal se comporta erráticamente después de haber cerrado el programa. Esto se debe a que la función *endwin* debe ser llamada antes de cerrar el programa. Instale un manejador de señal que llame a *endwin* para evitar este problema.

Requisitos del sistema

En Ubuntu, para poder usar la biblioteca ncurses, usted debe instalarla mediante el siguiente comando:

sudo apt-get install libncurses5-dev

Información útil

man fifo
man select_tut

Entrega

El proyecto puede ser entregado hasta el 9/2/2016 (martes de la semana 8) a las 11:59 p. m. La entrega consistirá de un archivo comprimido en formato *tar.gz* con los archivos fuente de su programa, dentro de un directorio. No incluya el ejecutable ni los archivos de objeto. Nombre el archivo comprimido con algo que distinga a su equipo, por ejemplo, proyectoApellido.tar.gz.

La entrega será por la plataforma Moodle del curso.