Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

IC-4700 Lenguajes de Programación

Tarea Programada 1: Mensajería en C

Prof. Andréi Fuentes L.

Estudiantes:

Susana Cob García 2017136954

Alejandro Tapia Álvarez 2017116952

22/03/19

Contenidos

I. Descripción del problema

El programa consiste en un sistema de mensajería implementado en el lenguaje de programación C.

Restricciones generales

- El mensaje de un cliente a otro debe ser recibido por el servidor y enviado por el mismo.
- Se debe utilizar la función fork para el envío y la recepción de mensajes, simultáneamente.
- Se deben utilizar colores para diferenciar los mensajes.
- El puerto y el IP del servidor deben ser especificados en un archivo de configuración, al igual que el puerto del cliente. Cada cliente tiene un archivo de configuración. El cliente puede cambiar su puerto.
- Se debe obtener la IP del usuario automáticamente.

II. Diseño de la solución

En este proyecto se utilizan sockets UDP. Los sockets TCP tienen conflictos con la función fork, ya que los sockets de los clientes sólo reconocen a sockets anteriores al mismo, por lo tanto, si se intentara enviar un mensaje de un cliente A a un cliente B que se conectó después, no sería posible, pues A no reconoce la existencia de B.

Fork

Fork es un mecanismo por el cual un proceso (padre) crea otro proceso (hijo). Los nuevos procesos actúan independientemente y no comparten memoria con el padre, es decir, crean copias de las variables del proceso padre.

En cuanto al valor devuelto por el fork:

a) -1 si se produce algún error en la creación y ejecución del fork.
b) 0 si no hay error y estamos en el proceso hijo.
c) PID (identificador del hijo) si no hay error y estamos en el proceso padre.

El fork se utiliza tanto en el servidor como en el cliente. En el cliente para crear procesos que envíen y reciban mensajes simultáneamente. En el servidor para manejar las acciones de cada cliente (envío y recepción de mensajes).

Servidor

Para el servidor se utilizan dos sockets con diferentes puertos, definidos previamente. El primer socket es para recibir a los usuarios que se conectan y así registrar sus datos en:

a) Una estructura que guarda el puerto, el IP y el nombre del cliente (el puerto se obtiene automáticamente con la estructura sockaddr_in).
b) Un char pointer que guarda el nombre del cliente.

Como estamos hablando de un servidor multi-cliente con comunicación entre clientes, las variables anteriores de cada cliente deben ser visibles para todos. Y como cada cliente es manejado mediante un proceso independiente con memoria independiente (fork), estas variables deben ser compartidas.

Para lograrlo, utilizamos Shared Memory (memoria compartida). Shared Memory comparte únicamente punteros a otros tipos de variables que no sean punteros. Por lo tanto, tenemos un arreglo (pointer) a las estructuras de cada cliente. Como las estructuras guardan un char pointer, vamos a compartir un char pointer con los nombres de todos los clientes (el nombre de un cliente va a estar formado por máximo 16 caracteres). La estructura X guarda, en su atributo de nombre, la dirección donde empieza el nombre de X cliente en el char pointer compartido. En el Shared Memory también se comparte un int pointer que guarda la cantidad de clientes que han ingresado al programa.

El segundo socket se encarga de recibir y enviar mensajes. Este socket se encontrará dentro del fork.

Se utilizan dos sockets porque hay un problema cuando un servidor UDP intercambia varios datagramas con el cliente. El problema es que el único número de puerto que el cliente conoce para el servidor es un puerto conocido por varios clientes. El cliente envía el primer datagrama como solicitud a ese puerto, pero ¿cómo distingue el servidor entre los datagramas posteriores de ese cliente y las nuevas solicitudes? La solución a este problema es que el servidor cree un nuevo socket para manejar los intercambios de datagramas de cada cliente y el otro socket se encarga de recibir nuevas solicitudes.

Cuando el servidor envía un mensaje, busca en el array con las estructuras de los clientes, el nombre de usuario que coincide con el destinatario y cuando lo encuentra, extrae los demás datos de la estructura para crear un nuevo struct sockaddr y enviar el mensaje. También, busca el nombre del remitente, igualmente lo busca en el array con las estructuras, pero lo busca por puerto.

Cuando un cliente se desconecta, la estructura del cliente se incializa de nuevo, en cero (en el array compartido).

Cliente

Cada cliente tiene un archivo de configuración donde se especifica el IP del servidor, los dos puertos del mismo y el puerto del cliente, con el siguiente formato:

IPS:*Puerto1S*:*Puerto2S*

Puerto:*Numero*

Antes de que el cliente comience el respectivo proceso con el servidor, se le pide el nombre de usuario y se le da la opción de cambiar el puerto. Una vez que se crea el socket del cliente, este envía un mensaje al servidor con el nombre de usuario (enviado al primero socket del servidor).

Una vez enviado el nombre de usuario, se ejecuta la función fork, donde el proceso hijo recibe mensajes y el proceso padre envía mensajes.

El formato para enviar mensajes es:

nombreUsuario:*mensaje*

El cliente recibe mensajes con el siguiente formato:

From *nombreUsuarioRemitente*: *mensaje*

Los mensajes que recibe el cliente son de color verde.

III. Referencias Bibliográficas

1. Stevens, W.; Fenner, B.; Rudoff, A. (2003). *Concurrent UDP Servers*. Recuperado de: http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch22lev1sec7.html
2. (2011). *Help on binding two sockets to two ports on ONE server [Socket Programming in UNIX]*. Recuperado de: http://forums.codeguru.com/showthread.php?514489-Help-on-binding-two-sockets-to-two-ports-on-ONE-server-Socket-Programming-in-UNIX
3. *UDP Server-Client implementation in C*. Recuperado de: https://www.geeksforgeeks.org/udp-server-client-implementation-c/
4. (2016). *Programación en C/Sockets*. Recuperado de: https://stackoverflow.com/questions/35626353/difference-between-a-tcp-socket-and-a-connected-udp-socket
5. (2018). *Programación en C/Sockets*. Recuperado de: https://es.wikibooks.org/wiki/Programación_en_C/Sockets
6. *fork() in C*. Recuperado de: https://www.geeksforgeeks.org/fork-system-call/
7. Delis, A. (2017). *Internetworking with Sockets*. Recuperado de: http://cgi.di.uoa.gr/~ad/k24/set006.pdf
8. Hall, B. (2016). *Beej's Guide to Network Programming Using Internet Sockets.* Recuperado de: http://beej.us/guide/bgnet/html/single/bgnet.html
9. (2013). *Trying to write to an int in shared memory (using mmap) with a child process.* Recuperado de: https://stackoverflow.com/questions/19672778/trying-to-write-to-an-int-in-shared-memory-using-mmap-with-a-child-process
10. (2009). *Is there a way for multiple processes to share a listening socket?* Recuperado de: https://stackoverflow.com/questions/670891/is-there-a-way-for-multiple-processes-to-share-a-listening-socket
11. (2010). *File Descriptor Sharing between Parent and forked Children*. Recuperado de: https://stackoverflow.com/questions/3952362/file-descriptor-sharing-between-parent-and-forked-children
12. Universidad de las Palmas de Gran Canaria. *Lección 7: Creacion de Procesos. Fork.* Recuperado de: http://sopa.dis.ulpgc.es/ii-dso/leclinux/procesos/fork/LEC7_FORK.pdf