Tarea Programada 1

Lenguajes de Programación

Prof. Andrei Fuentes Leiva

Kevin Sem Piedra Matamoros
2015069024

Daniel Rodríguez
2016136187

Gabriel Omar Piedra Camacho
2016078985

22 de Marzo del 2019

Introducción

Se desarrollara un sistema de chat que utiliza Berkeley sockets para la comunicación entre el cliente y el servidor. El servidor podrá aceptar y manejar varias conexiones de clientes en paralelo. Además de esto, los clientes pueden enviar y recibir mensajes al mismo tiempo, para lo que usarán fork(). El servidor mantendrá control de los clientes conectados y nombres de usuario disponibles. Una vez conectado, un cliente decidirá a quién quiere enviar un mensaje y puede escribirlo, el server se encargará de enviar el mensaje al usuario correcto.

El programa se desarrolló para Linux, con el compilador **GCC** (GNU Compiler Collection) versión **8.1.0**, y su implementación de la librería estándar glibc (GNU C Library, que es incluida automáticamente). Se utiliza la el estándar C11 del lenguaje C (mediante la opción `-std=c11` del programa gcc).

Descripción de los archivos

Archivos de código fuente:

- IOHandler.h: Métodos y funciones para leer y escribir archivos de configuración.
- IOHandler.c: Implementación de IOHandler.h
- client_main.c: Implementación del programa cliente.
- server.h: Funciones, métodos, y declaraciones utilizadas en la implementación del server.
- server.c: Implementación de "server.h".
- server_main.c: Código principal del server, que hace uso de "server.h".
- makefile: Archivo MAKEFILE para ser usado con GNU make, para construir los ejecutables.

Otros archivos:

- client.conf: El archivo de configuración para los clientes.
- server.conf: El archivo de configuración para el server.
- .gitignore: Usado para hacer que Git ignore ciertos archivos (e.g.: los ejecutables).
- "Documentación Proyecto 1 Lenguajes.pdf": Este archivo de documentación del proyecto.

Compilación/makefile

Los ejecutables pueden ser generados mediante el uso del archivo makefile. Éste provee las siguientes tasks:

- make server: Compila “server”
- make client: Compila “client”

- make build: Compila ambos, “server” y “client”. Esta es la opción por default al llamar a make.
- make clean: Intenta eliminar el archivo “server”, y el archivo “client”.

Ejemplo:

```
sempiedram@Kevin-Laptop ~/Projects/TEC/PL119A_b
 $ make
gcc -Wall -Wextra -std=c11 -o server IOHandler.c server_main.c server.c
gcc -Wall -Wextra -std=c11 -o client IOHandler.c client_main.c
```

Uso del chat

Para usar el chat se debe ejecutar el server (“./server”). Cuando el server esté listo para aceptar conexiones, los clientes se pueden ejecutar con “./client”.

En el siguiente link se puede encontrar un video demostrando el uso del chat: https://drive.google.com/open?id=1IoKfmLHjCagC4pNUUdGu01YBOmftK6gF

Ejemplos:

```
sempiedram@Kevin-Laptop ~/Projects/TEC/PL119A_b
 $ ./server
INFO: Server bound to port "50777"

INFO: Waiting with select() for any client activity.
INFO: accept()'ing client connection.
INFO: New client connected with socket FD: 4.

INFO: Waiting with select() for any client activity.
```

```
sempiedram@Kevin-Laptop ~/Projects/TEC/PL119A_b
 $ ./client
INFO: Getting the ip address for "localhost".
INFO: Server address for "localhost": 127.0.0.1
Ingrese su nombre de usuario: kevin
DEBUG: Sending username: "#kevin".

Formato para enviar un mensaje: Usuario:Mensaje
Para salir, escribir "$S".

gabriel:Hola!
                                                    gabriel:Pura vida?!
gabriel:Si!
                                                    gabriel:Ok, Adios!
gabriel:OK!
```

Diseño

El sistema será diseñado para el sistema operativo Linux en el lenguaje de programación C. El sistema consistirá de dos programas independientes: el programa del cliente y el programa del server. El server se abrirá en un puerto especificado y los clientes especificaran a qué puerto se quieren conectar a través de un archivo de configuración aparte.

El cliente usará la función fork() para poder enviar mensajes y recibirlos al mismo tiempo. Al usar fork() uno de los procesos se enfocara en escuchar cualquier mensaje que pueda venir del server, mientras que el otro proceso escuchara el input del usuario para enviarlo al server.

La función fork(), después de llamada, crea un nuevo proceso que correrá de desde donde se llamó el fork() en adelante. Este nuevo proceso es conocido como el proceso hijo, y el proceso padre también ejecutará el código que sigue después de la llamada del fork(). Para poder identificar cual es el proceso hijo nada más se debe ver el valor que retorna el fork(): si es 0 significa que es el hijo, y si es un número positivo, es el padre.

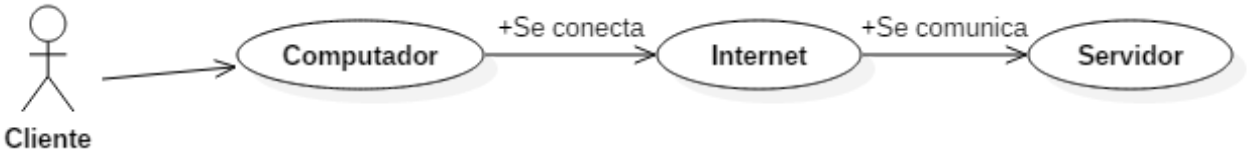
El cliente tendrá un archivo de configuración llamado “client.conf”, donde se especificara en qué puerto, y en qué dirección IP se encuentra el server.

Los mensajes que se reciben de otros usuarios serán formateados de manera diferente. Cuando se recibe un mensaje de otro usuario este se colocará al lado derecho de la terminal, y además se mostrarán en color verde para diferenciarlos de los mensajes que se escriben.

El servidor mantendrá control de todos los usuarios que se conectan a este mismo. El servidor se asegurará que no se puedan conectar dos usuarios con el mismo nombre al mismo tiempo, y se asegurará que una vez que algún cliente salga del sistema ese nombre de usuario se libere y pueda ser usado por otros nuevos usuarios.

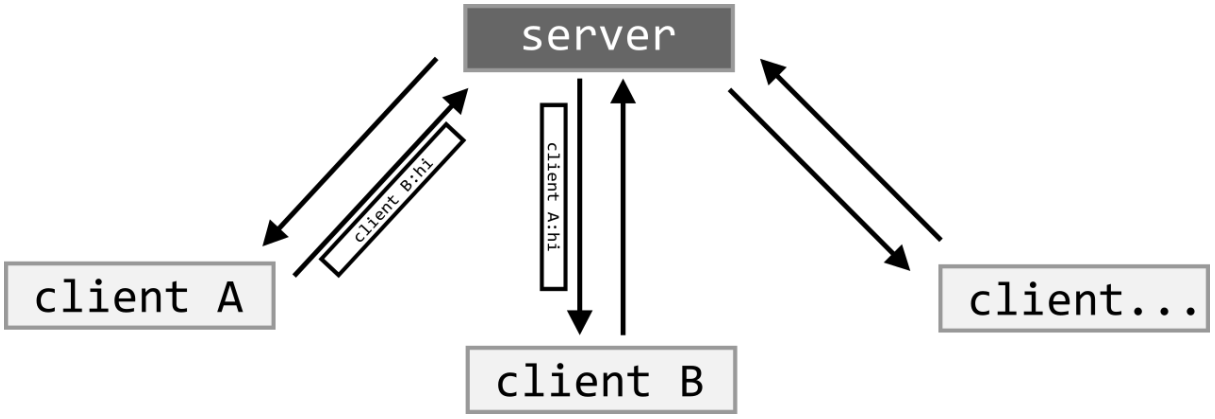
En el servidor, para poder manejar los múltiples sockets de los clientes abiertos, se utilizó el método select(). El método select() permite esperar hasta que haya algún tipo de actividad en un conjunto de sockets, y después determinar cuales sockets tenían actividad disponible.

Diagrama de Arquitectura



Representación de múltiples clientes conectados:

Aquí se muestran tres clientes conectados al mismo tiempo a un server, y el envío de un mensaje desde el cliente A al cliente B.



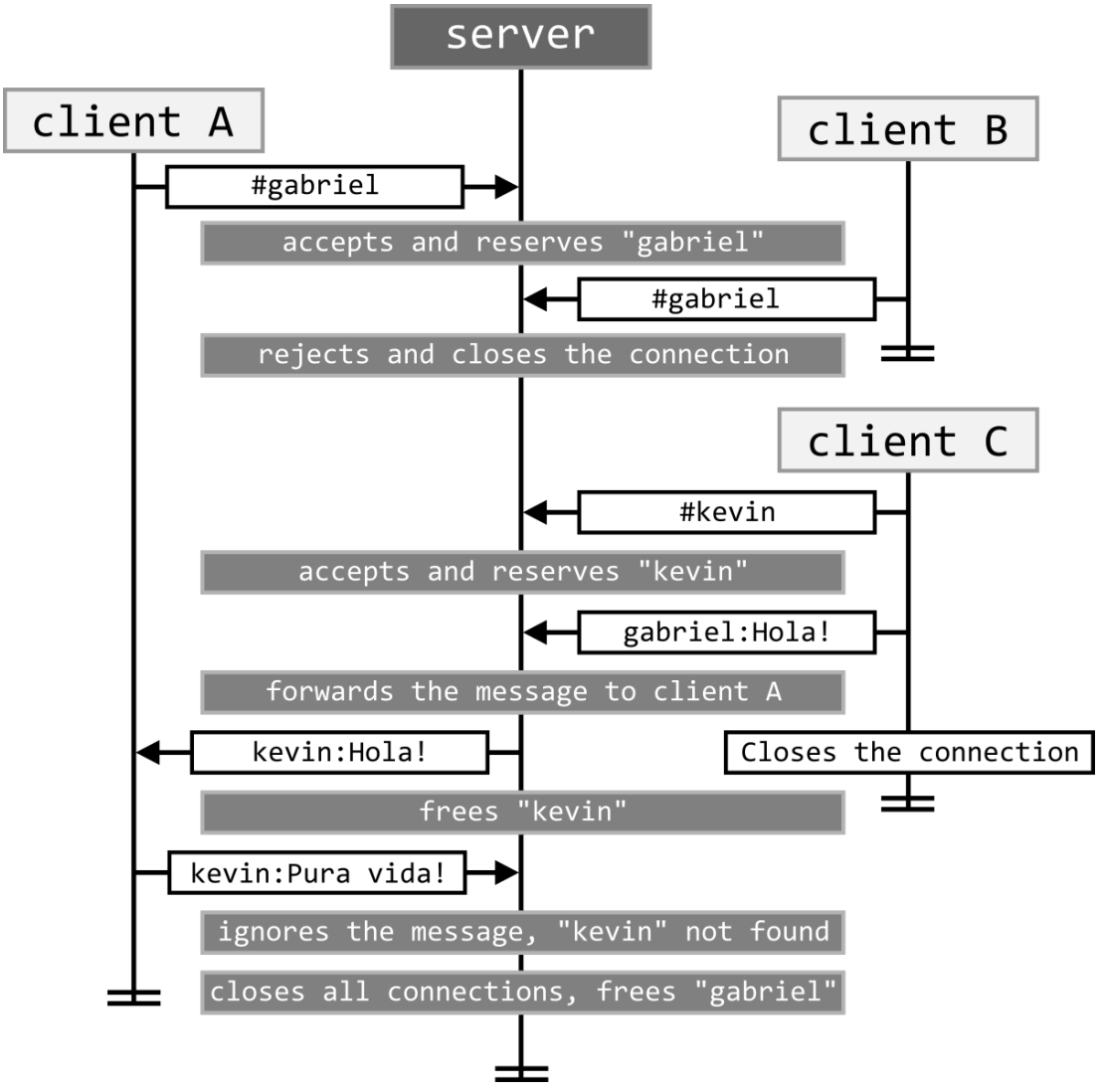
Ejemplo de una posible sesión

En el siguiente diagrama se muestra una posible sesión de uso del chat. Primero se conecta el cliente A al server, y solicita el username “gabriel”. El server acepta la solicitud y reserva el nombre. Luego el cliente B se conecta y solicita también el nombre de usuario “gabriel”, pero este ya está en uso, así que el server rechaza y cierra esa conexión.

Después se conecta el cliente C y solicita el username “kevin”, y el servidor lo acepta. Este cliente manda el mensaje “Hola!” al cliente con el nombre “gabriel” asignado. El server entonces le informa al cliente A que ha recibido un mensaje del cliente con nombre “kevin”. Luego el cliente C se desconecta, y el server libera el nombre “kevin”, por lo que un nuevo cliente podría utilizarlo.

El cliente A entonces intenta mandarle un mensaje al cliente con nombre “kevin”, pero no hay ningún cliente conectado que tenga ese nombre, así que el server ignora este mensaje.

Finalmente, el server cierra todas las conexiones y termina su ejecución.



Referencias

A. M. (2016, September 17). Adding Color to Your Output From C. Tomado de: http://web.theurbanpenguin.com/adding-color-to-your-output-from-c/

Fork(2). (2015, September 15). Tomado de: http://man7.org/linux/man-pages/man2/fork.2.html

Robert I. Pitts. (s.f). Intro to File Input/Output in C. Tomado de: https://www.cs.bu.edu/teaching/c/file-io/intro/

Programming simplified. (s.f.). C read file program. Tomado de: https://www.programmingsimplified.com/c-program-read-file