

# **Tecnológico de Costa Rica**

Escuela de Computación

IC-4700 Lenguajes de programación

## **Documentación - Proyecto #1:**

Programación Imperativa

### **Estudiantes:**

Jorge Arturo Chavarría Villarevia- 2023157488

Murillo Chaves Felipe Javier - 2023083005

Hidalgo Sánchez Fernando Andrés - 2022218688

Grupo 02

### **Profesor:**

Bryan Tomas Hernandez Sibaja

I Semestre

2025

## Índice

Enlace a GitHub.....	2
Descripción.....	2
Diseño del programa.....	2
Manual de Usuario.....	5

## Enlace a GitHub

<https://github.com/ArturoChV10/proyectoLenguajes.git>

---

## Descripción

El proyecto #1 para el curso de Lenguajes de programación tiene como objetivo familiarizar a los estudiantes con el desarrollo de programas en C, mediante la creación de un programa de mensajería. Dentro de este sistema se va a tener un servidor central que almacene toda la información de los usuarios para luego cuando el usuario A quiera enviar un mensaje al usuario B, primero este mensaje pasará por el servidor central antes de llegar hasta el usuario B.

Los usuarios se deben registrar en el servidor central, al hacer esto se obtiene su IP de forma automática. Además pueden enviar mensajes a sus contactos, especificando su nombre de usuario y el mensaje que quieren enviar, dichos mensajes como se dijo antes no son enviados directamente al otro usuario, sino que tienen que pasar por el servidor central antes. Por último los usuarios pueden recibir mensajes de otros usuarios.

---

## Diseño del programa

El sistema fue creado en el lenguaje de programación C++ y por medio de una terminal de Linux (usando Ubuntu) se puede acceder y poner a funcionar el proyecto.

El sistema funciona de la siguiente manera: Se implementa un Servidor socket:

Cuando se desea manejar un servidor socket, lo primero sería definir una dirección para el mismo, y es durante este proceso que utilizamos la función socket, la cual nos pide especificar el protocolo (AF\_INET para IPv4) y el tipo de socket a utilizar (SOCK\_STREAM para un servidor TCP).

```
int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
```

**Proyecto - Programación Imperativa**

Luego se define una dirección sobre la cual el servidor estará siendo ejecutado, de manera que podamos acceder a este en todo momento, en este momento se asocia el protocolo a utilizar, se toma un valor entero que será utilizado como puerto y se convierte de byte de máquina a byte de red (htons(8080)), y por último se especifica que el servidor no esté asociado a ninguna IP específica, sino que esté disponible para cualquiera (INADDR\_ANY)

```
sockaddr_in serverAddress;  
serverAddress.sin_family = AF_INET;  
serverAddress.sin_port = htons(8080);  
  
serverAddress.sin_addr.s_addr = INADDR_ANY;
```

Luego de haber configurado las especificaciones del servidor pasaremos a ligar el entero que obtuvimos inicialmente (el cual hace referencia a la dirección del servidor) al servidor configurado anteriormente.

```
bind(serverSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress));
```

El próximo paso a seguir corresponde a hacer que el servidor socket "escuche" conexiones, de manera que si un cliente desea acceder al servidor este lo encuentre sin problemas, para esto utilizamos la función listen(), la cual recibe el servidor como tal y la cantidad de usuarios activos.

```
listen(serverSocket, 5);
```

Por último falta aceptar a los clientes que deseen acceder al sistema, para esto se implementó un ciclo while(true) que permite aceptar a tantos usuarios como se desee (siempre y cuando no desborde el sistema), para aceptar clientes se utiliza la instrucción accept, la cual recibe tres parámetros diferentes, el primero corresponde al servidor, el segundo nos permite obtener la dirección del cliente mediante el uso de punteros, pero como se aprecia este espacio está ocupado nullptr y esto se debe a un problema con el acceso a las IP's de los clientes, lo cual generaba muchos problemas, por último se tiene un espacio para el tamaño de la dirección, pero esto no es necesario para nuestro caso.

```
int clientSocket = accept(serverSocket, nullptr, nullptr);
```

Externo a la definición del servidor también existe una instrucción para recibir datos de un cliente, dicha instrucción es recv(), la cual manejará un arreglo de caracteres para almacenar el mensaje recibido.

```
char buffer[1024] = {0};  
recv(clientSocket, buffer, sizeof(buffer), 0);  
cout << "Message from client: " << buffer << endl;
```

Finalmente la instrucción para cerrar el servidor corresponde a la siguiente.

```
close(serverSocket);
```

#### Conexión de cliente:

Para la conexión de cliente se utilizaron diversas instrucciones, muchas de estas instrucciones repiten parámetros con las del servidor, por lo que dicha explicación será omitida.

Para crear un cliente se utiliza nuevamente `socket()`, este cliente será almacenado en una variable entera.

```
int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
```

Luego se debe definir la dirección y características del servidor al que realizará la conexión.

```
sockaddr_in serverAddress;  
serverAddress.sin_family = AF_INET;  
serverAddress.sin_port = htons(8080);
```

Para conectar el cliente con el servidor, se utilizará una instrucción similar al `bind()` de los servidores, esta instrucción se define como `connect()`

```
connect(clientSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress));
```

Para finalizar la conexión entre un servidor y un cliente se utiliza la siguiente instrucción.

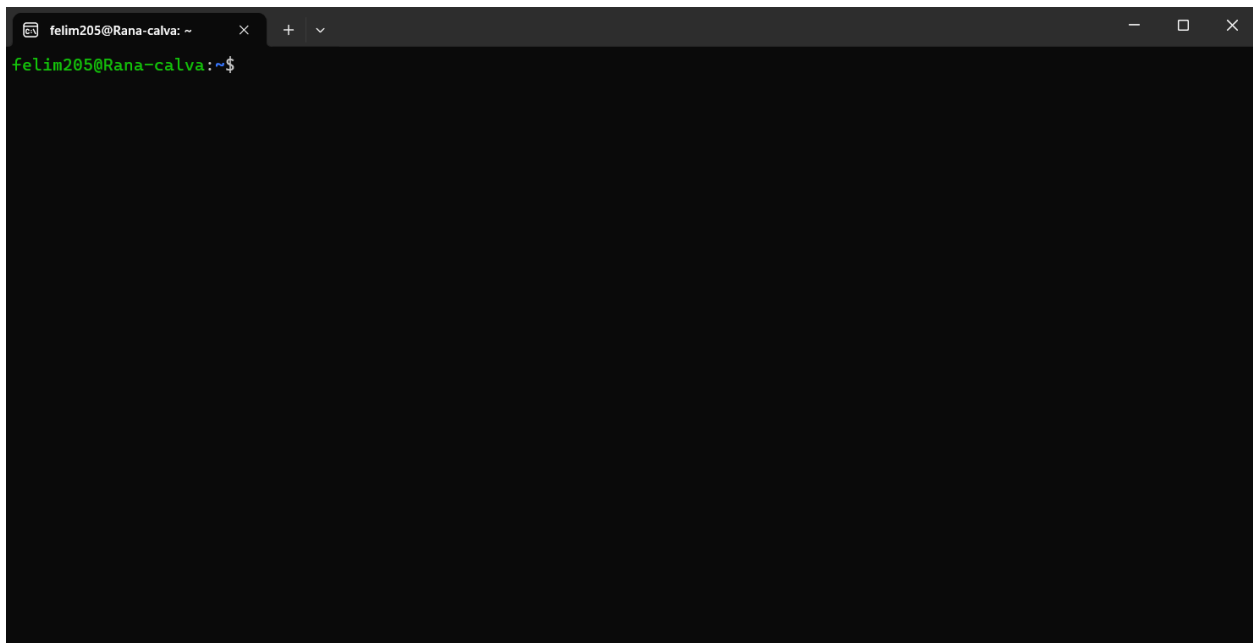
```
close(clientSocket);
```

## Manual de Usuario

En esta sección se presenta el manual de usuario para la aplicación desarrollada de mensajería. Aquí se explica brevemente la función del sistema y sus componentes, con el objetivo de facilitar el uso de la aplicación. Este manual servirá como guía de referencia para facilitar su uso. Aunque el documento cumple con un aspecto formal, se ha redactado para la comodidad del usuario final.

Primero se van a descargar todos los componentes necesarios, serían los siguientes:

- Bajar proyecto del github: <https://github.com/ArturoChV10/proyectoLenguajes.git>
- Utilizar una máquina virtual, partición o el método que le sea más cómodo, esto porque se ocupa correr el proyecto en un entorno de Linux. Para nuestro caso por medio de Ubuntu, descargando la aplicación se puede acceder a una terminal de Linux:



Una vez abierta esta terminal se va a ejecutar el siguiente comando:

**cd /ruta-del-proyecto/**

Ejemplo:

**cd /Users/<nombreUsuario>/<Documentos>/proyectoLenguajes**

En caso de no servir probar con el siguiente:

**cd /mnt/c/Users/<nombreUsuario>/<Documentos>/proyectoLenguajes**

**ATENCIÓN:** Esta es la ruta de uno de los desarrolladores pero el usuario deberá colocar la ruta donde descargo el proyecto del github.

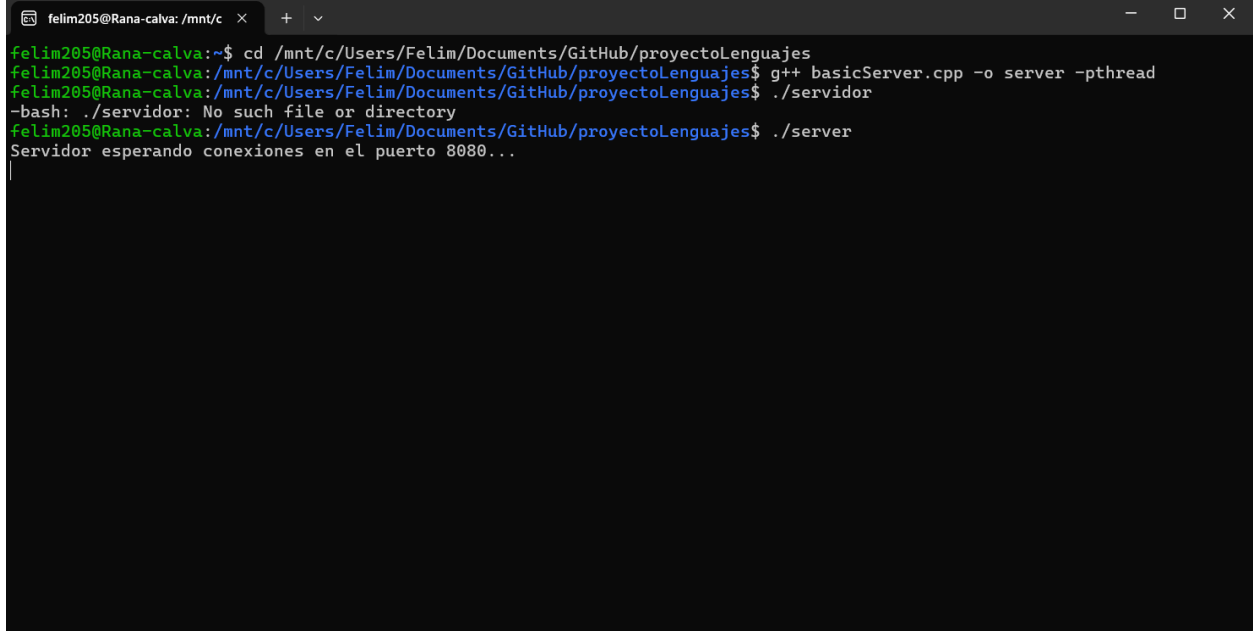
Una vez dentro del proyecto se ejecutará el siguiente comando (**nota:** no va a salir nada, es para inicializar):

**g++ basicServer.cpp -o server -pthread**

Luego se inicializa el servidor con el siguiente comando:

**./server**

Debera verse asi:



```
felim205@Rana-calva: /mnt/c
felim205@Rana-calva:~$ cd /mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes
felim205@Rana-calva:/mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes$ g++ basicServer.cpp -o server -pthread
felim205@Rana-calva:/mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes$ ./servidor
-bash: ./servidor: No such file or directory
felim205@Rana-calva:/mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes$ ./server
Servidor esperando conexiones en el puerto 8080...
```

Siguiente paso, se van a abrir 2 terminales aparte donde primero nos vamos a dirigir a la ruta del proyecto y luego ejecutar el siguiente comando:

**g++ clientRequest.cpp -o cliente -pthread**

despues

**./cliente**

Deberá salir lo siguiente:

```
felim205@Rana-calva: /mnt/c  x  +  v
felim205@Rana-calva:~$ cd /mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes
felim205@Rana-calva:/mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes$ g++ clientRequest.cpp -o cliente -pthread
felim205@Rana-calva:/mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes$ ./cliente
Conectado al servidor
Desea crear una cuenta nueva? (y/n)|
```

Se va a crear un usuario para poder enviar mensajes, después se le va a pedir que haga login y luego lo deja seleccionar la persona a la cual le va a enviar un mensaje, debe colocar el usuario de esta o le puede enviar mensajes al administrador colocando “admin”.

```
felim205@Rana-calva: /mnt/c  x  +  v
felim205@Rana-calva:~$ cd /mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes
felim205@Rana-calva:/mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes$ g++ basicServer.cpp -o server -pthread
felim205@Rana-calva:/mnt/c/Users/Felim/Documents/GitHub/proyectoLenguajes$ ./server
Servidor esperando conexiones en el puerto 8080...
Cliente conectado!
Mensaje del cliente: Daniel TO admin: 2
Cliente conectado!
Cliente desconectado.
Cliente conectado!
Mensaje del cliente: RanaCalva TO GaboBaVi: Grop grop grop...
Mensaje del cliente: GaboBaVi TO RanaCalva: Alo

felim205@Rana-calva: /mnt/c  x  +  v
Conectado al servidor
Desea crear una cuenta nueva? (y/n)y
Cree un nombre de usuario: GaboBaVi
Cree una contraseña: Danielito2
Usuario registrado exitosamente.
Usuario: GaboBaVi Contraseña: Danielito2
Ingrese su nombre de usuario:
Ingrese su contraseña: Danielito2
Nombre de usuario o contraseña incorrectos.

Ingrese su nombre de usuario: GaboBaVi

Ingrese su contraseña: Danielito2
Bienvenido, GaboBaVi!
Receptor del mensaje:
Mensaje recibido de RanaCalva : Grop grop grop...
RanaCalva
GaboBaVi: Alo
Receptor del mensaje:

felim205@Rana-calva: /mnt/c  x  +  v
La contraseña debe ser de al menos 8 caracteres, contener una mayuscula y un digito
Cree una contraseña: calvaranA3
Usuario registrado exitosamente.
Usuario: RanaCalva Contraseña: calvaranA3
Ingrese su nombre de usuario:
Ingrese su contraseña: RanaCalva
Nombre de usuario o contraseña incorrectos.

Ingrese su nombre de usuario: RanaCalva

Ingrese su contraseña: calvaranA3
Bienvenido, RanaCalva!
Receptor del mensaje: GaboBaVi
RanaCalva: Grop grop grop...
Receptor del mensaje:
Mensaje recibido de GaboBaVi : Alo
```

Y listo! Muchas gracias por usar nuestro sistema de mensajería, esperamos que esta guía sea útil para entender su funcionamiento.