**PROYECTO FINAL**

“Creación de un chat en tiempo real con sockets en lenguaje C”

**GRUPO:** 2CV15

**EQUIPO:** El Siuuu Team

**INTEGRANTES:**

Fischer Salazar César Eduardo

López García José Eduardo

Meza Vargas Brandon David

**PROFESOR:** Fabian Gaspar Medina

**FECHA:** 20/12/21

**índice**

[**Introducción** 3](#_Introducción)

**[Desarrollo](#_Desarrollo)** [4](#_Desarrollo)

**[Diagrama de flujo](#_Diagrama_de_flujo)** [4](#_Diagrama_de_flujo)

**[Programa del cliente](#_Programa_del_cliente)** [4](#_Programa_del_cliente)

**[Programa del servidor](#_Programa_del_servidor)** [6](#_Programa_del_servidor)

**[Sockets](#_Sockets)** [8](#_Sockets)

**[Sockets en c](#_Sockets_en_c)** [9](#_Sockets_en_c)

**[Programa realizado](#_Programa_realizado)** [11](#_Programa_realizado)

**[Programa del cliente](#_Programa_del_cliente)** [11](#_Programa_del_cliente)

**[Programa del servidor](#_Programa_del_servidor)** [16](#_Programa_del_servidor)

**[Pruebas de funcionamiento](#_Pruebas_de_funcionamiento)** [21](#_Pruebas_de_funcionamiento)

**[Instrucciones de ejecución](#_Instrucciones_de_ejecución)** [26](#_Instrucciones_de_ejecución)

**[Conclusiones](#_Conclusiones)** [27](#_Conclusiones)

**[Fischer Salazar César Eduardo](#_Fischer_Salazar_César)** [27](#_Fischer_Salazar_César)

**[López García José Eduardo](#_López_García_José)** [27](#_López_García_José)

**[Meza Vargas Brandon David](#_Meza_Vargas_Brandon)** [27](#_Meza_Vargas_Brandon)

**[Referencias](#_Referencias)** [29](#_Referencias)

# **Introducción**

A lo largo de la existencia de las redes, desde sus inicios era bastante importante conseguir que pudiera haber interacción a distancia entre dos personas que se encontraban bastante distanciadas una de otra, ya que antes era bastante complicado que se diera una comunicación en directo, cartas, faxs, telegramas, tardaban mucho en poder llegar a su destinatario, al igual que su respuesta.

Con el paso de los años, y de la mano con la evolución de la tecnología, ha sido posible para el ser humano poder contar con diferentes dispositivos y elementos que pueden conformar cualquier topología de red (anillo, estrella, malla, etc.) y que se cumpliera con dicho propósito.

Y es claro, se sabía dentro de esta área de debía existir alguna manera en las que dichas conversaciones fueran más fluidas, más rápidas, en tiempo real, alrededor del mundo, en donde se creara un ambiente agradable en el que se sintiera una buena convivencia y más conexiones de gente desde diferentes partes.

Hoy por hoy, podemos contar en nuestros ordenadores y celulares con sistemas de chat en línea que realizan esa interacción que se ha ido dando año con año entre diferentes plataformas, ha evolucionado y parece que irá teniendo un desarrollo y una evolución más allá de lo que tenemos al alcance de nuestras manos.

Llegados a este punto, con lo que hemos aprendido a lo largo de la materia, con prácticas realizadas en el sistema operativo Linux, y una que otra simulación, surgió un objetivo de poder tratar de recrear un sistema de chat parecido a uno con los que podemos contar actualmente; el cual se buscaba que tuviera una función bidireccional en el que se permitiera la interacción de usuarios de distintas partes, constando de un cliente y un servidor, donde sea posible verse reflejado en ambos puntos la cantidad de usuarios conectados, además de que contaran con un ID con el que se identificaran fácilmente al momento del envío de mensajes, e informar de su desconexión, todo esto a través de la programación en C, con apoyo del manual de Linux y el concepto de sockets crudos vistos en clase.

# **Desarrollo**

## **Diagrama de flujo**

### **Programa del cliente**

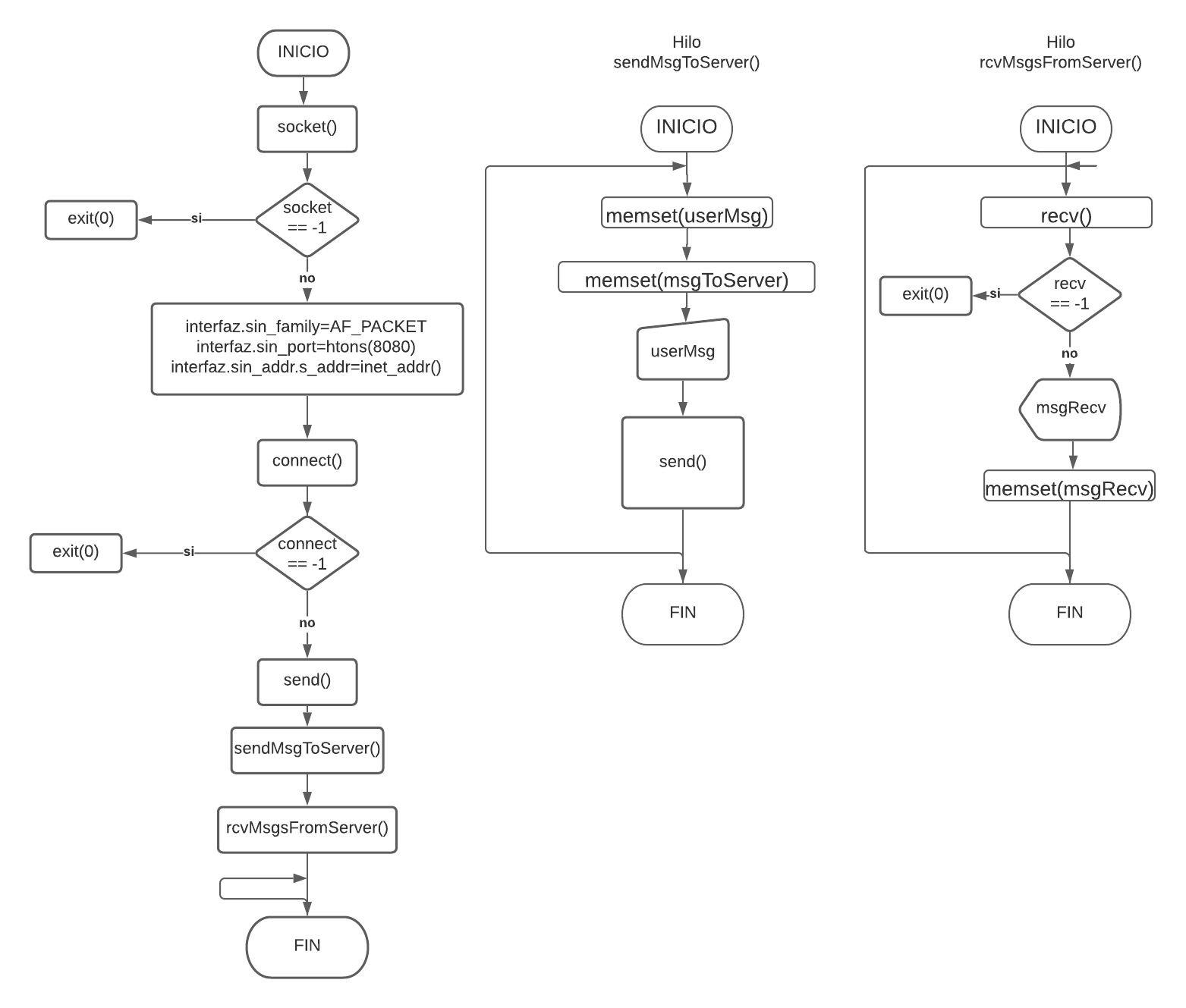


Imagen 1. main del programa cliente.

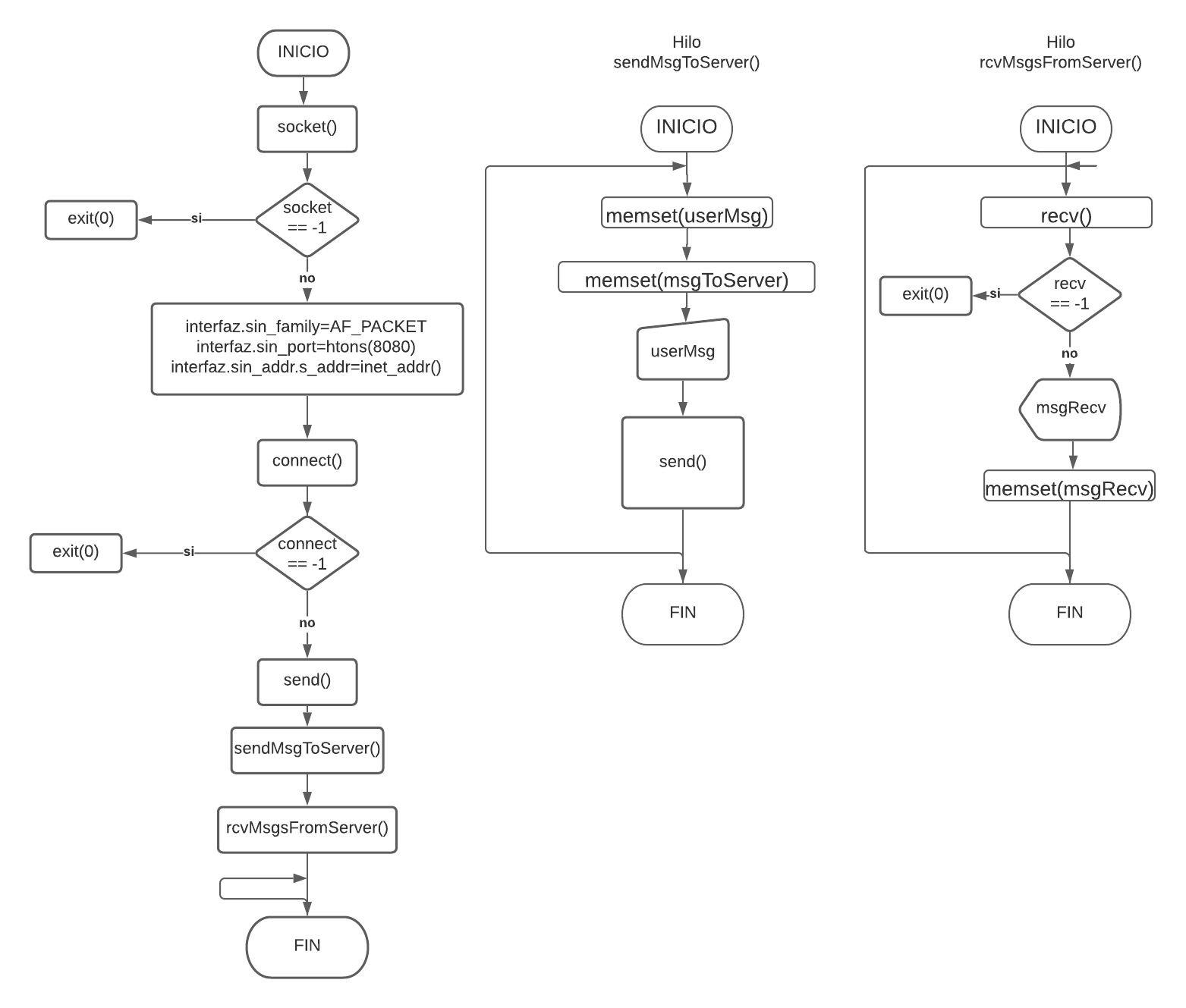


Imagen 2. Hilo que manda mensaje al servidor

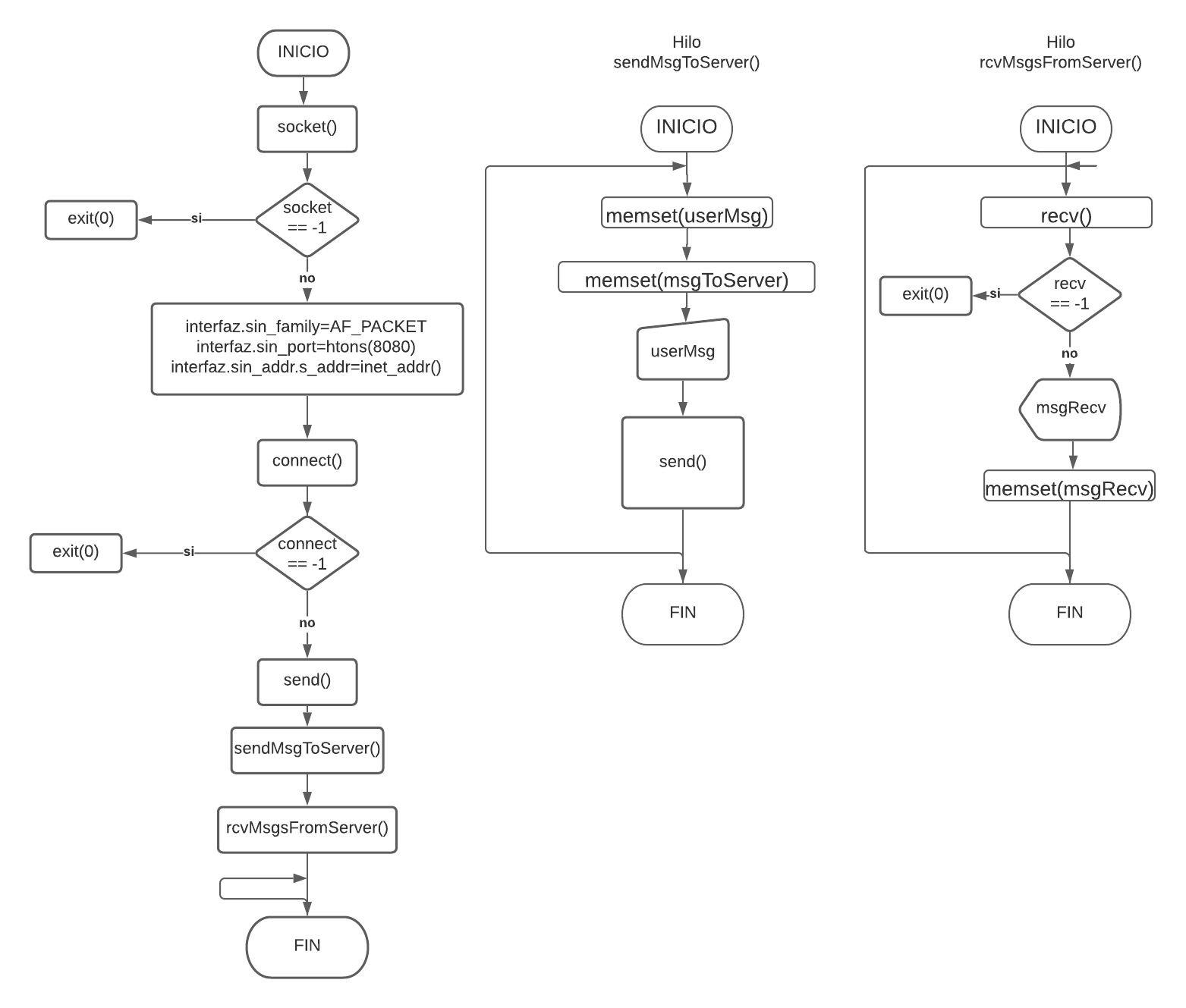


Imagen 3. Hilo que recibe mensajes del servidor.

### **Programa del servidor**

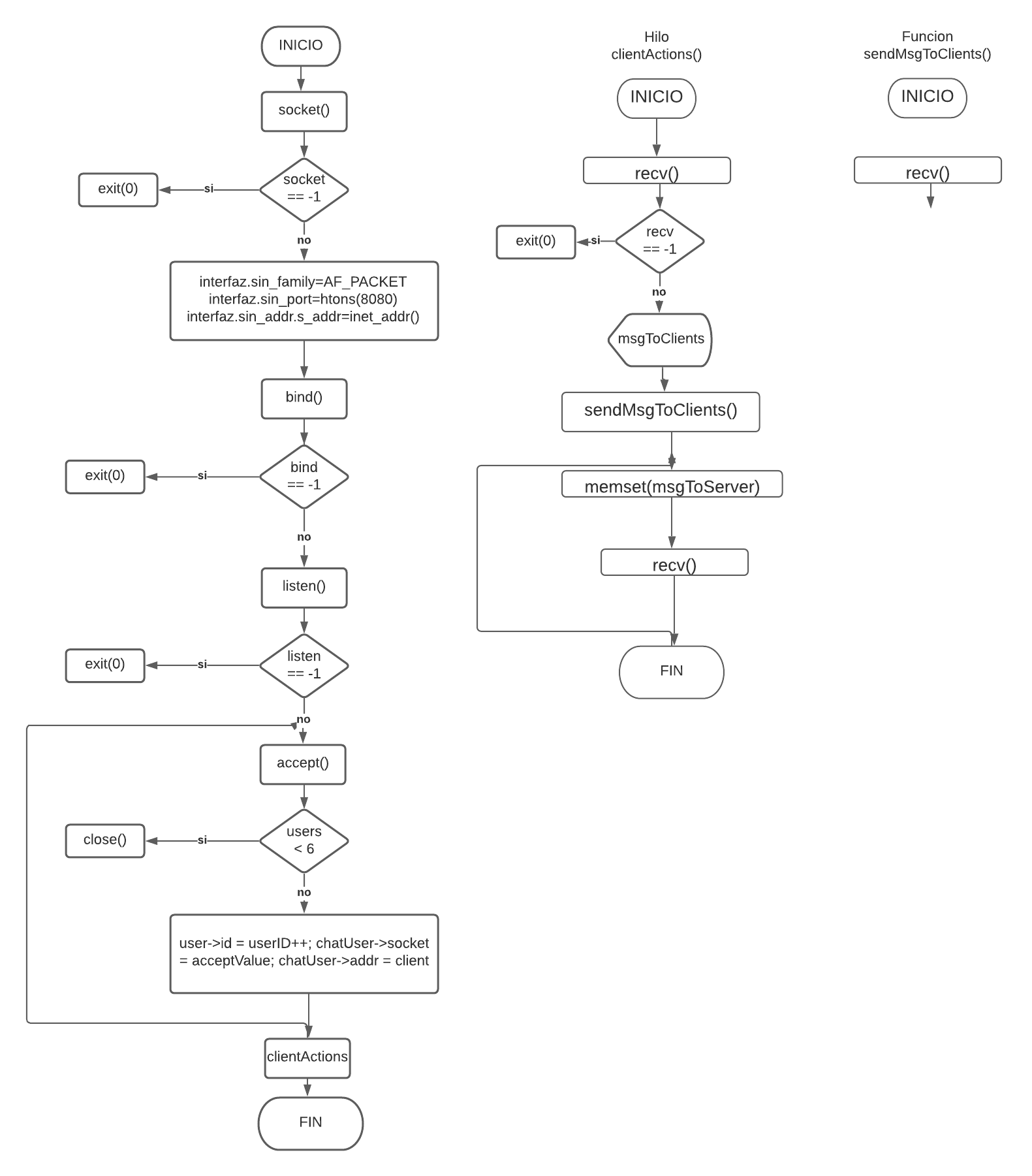


Imagen 4. main del servidor

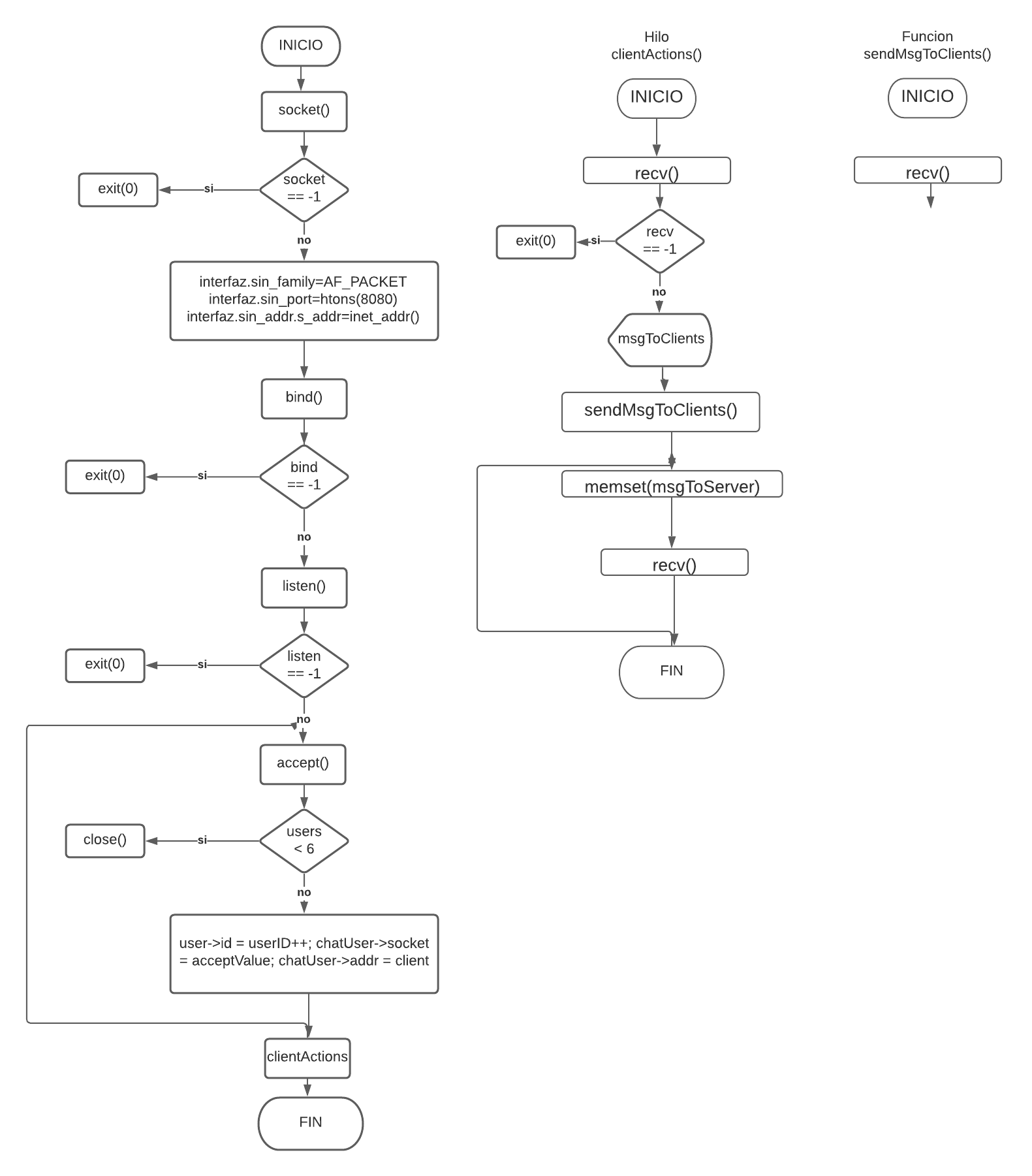


Imagen 5. Hilo que gestiona las acciones del cliente.

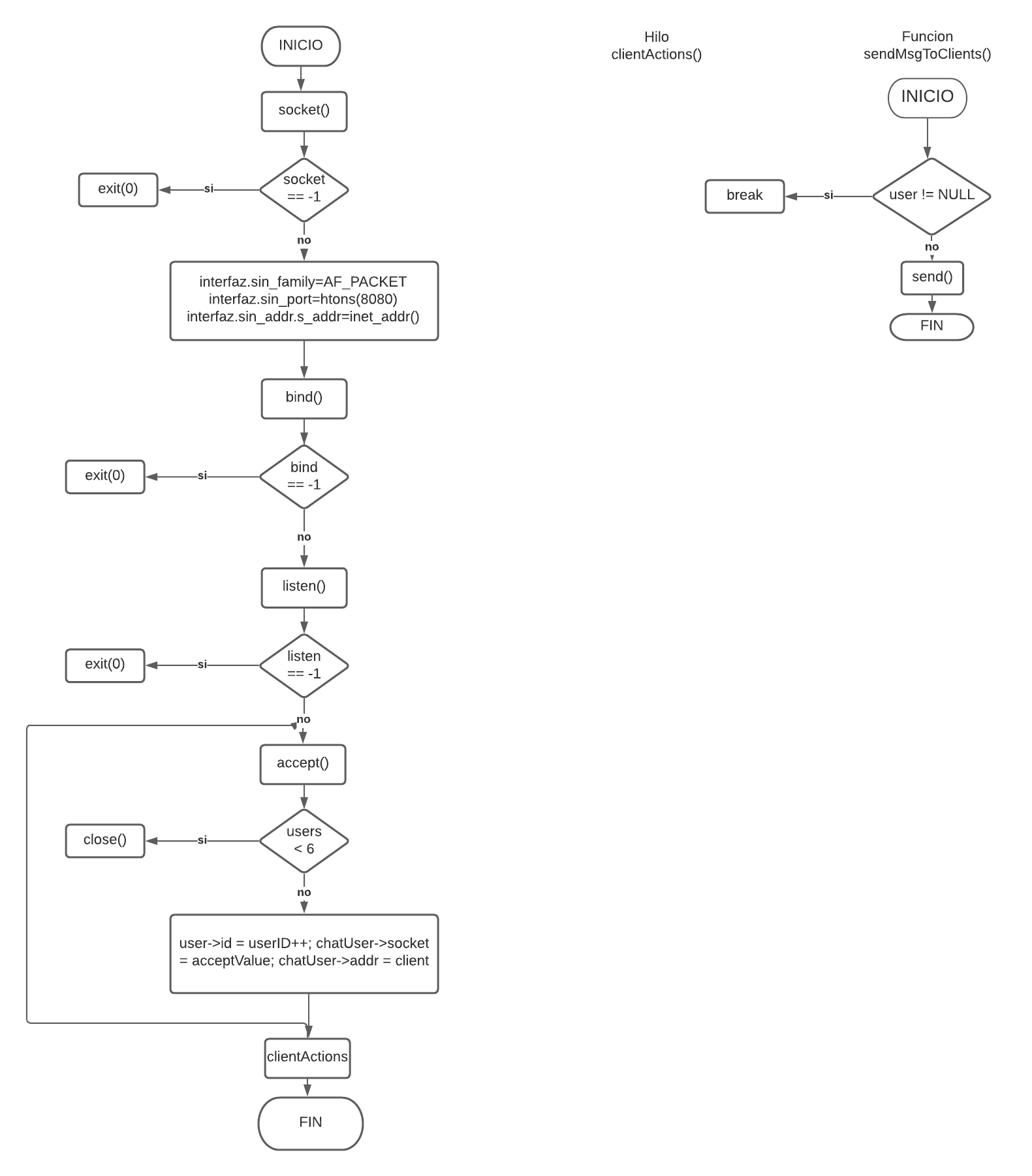


Imagen 6. Función que manda mensajes a los clientes conectados.

## **Sockets**

Un socket es un tipo de software que actúa como un punto final, que funciona estableciendo un enlace de comunicación de red bidireccional entre el extremo del servidor y el programa receptor del cliente. Estos sockets se realizan y movilizan junto con llamadas de función, y es capaz de simplificar el funcionamiento de un programa porque los programadores sólo tienen que preocuparse de manipular las funciones del socket para transportar los mensajes a través de la red correctamente.

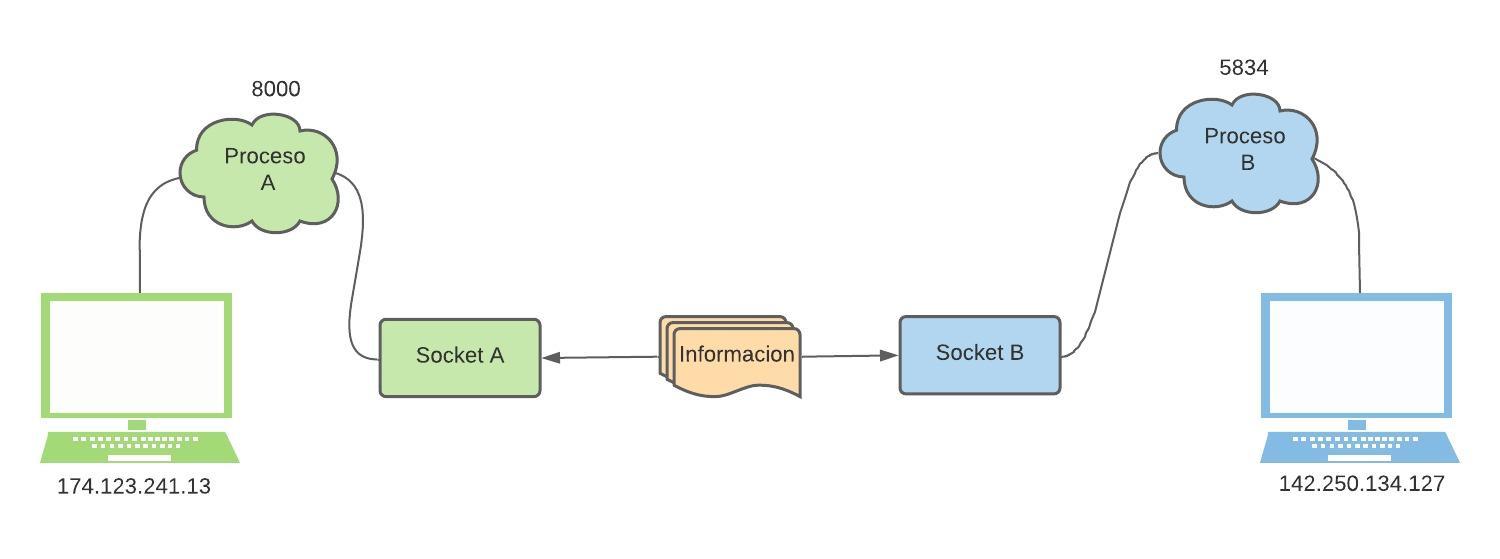


Imagen 7. Funcionamiento de sockets.

Para un modelo cliente-servidor orientado a la conexión, el socket en el proceso del servidor espera la petición de un cliente, aquí, el servidor necesita establecer una dirección que los clientes puedan usar para encontrar y conectarse al servidor. Cuando se establece una conexión con éxito, el servidor esperará a que los clientes soliciten un servicio. El intercambio de datos cliente-servidor tendrá lugar si el cliente se conecta al servidor a través del socket, entonces, el servidor responderá a la solicitud del cliente y enviará una respuesta.

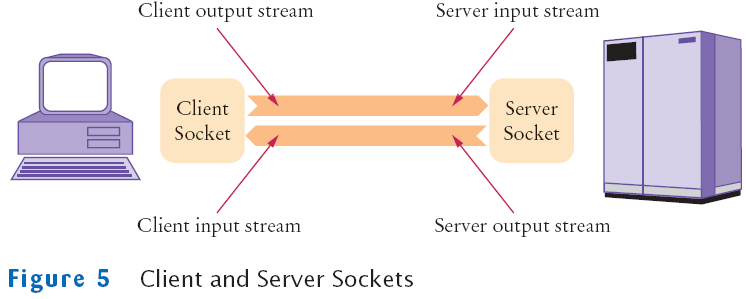


Imagen 8. Sockets de cliente y servidor

## **Sockets en c**

Dentro del lenguaje C, la palabra socket designa un concepto abstracto por el cual dos programas, que posiblemente se encuentren situados en computadoras distintas, pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.

Un socket es, un conector o enchufe con el que podremos conectarnos a ordenadores remotos o permitir que éstos se conecten al nuestro a través de la red. En realidad, un socket no es más que un descriptor de fichero un tanto especial. Dentro de lo que es UNIX todo es un fichero, así que, para enviar y recibir datos por la red, sólo tendremos que escribir y leer en un fichero un poco especial.

*Funciones*

1. Función socket()

Es una llamada al sistema, esta función devuelve un descriptor de archivo, al igual que la función open() al crear o abrir un archivo. Durante la llamada se reservan los recursos necesarios para el punto de comunicación, pero no se especifica nada con respecto a la dirección asociada al mismo.

int socket (int family, int type, int protocol);

1. Función bind()

Llamada al sistema que permite asignar una dirección a un socket existente. El sistema no atenderá a las conexiones de clientes, simplemente registra que cuando empiece a recibirlas le avisará a la aplicación. En esta llamada se debe indicar el número de servicio sobre el que se quiere atender.

int bind (int sockfd, const struct sockaddr \*myaddr, int addrlen);

1. Función listen()

Avisar al sistema operativo que comience a atender la conexión de red. El sistema registrará la conexión de cualquier cliente para transferirla a la aplicación cuando lo solicite. Si llegan conexiones de clientes más rápido de lo que la aplicación es capaz de atender, el sistema almacena en una cola las mismas y se podrán ir obteniendo luego.

int listen (int s, int backlog)

1. Función accept()

Pedir y aceptar las conexiones de clientes al sistema operativo. El sistema operativo entregará el siguiente cliente de la cola. Si no hay clientes, se quedará bloqueada hasta que algún cliente se conecte.

int connect(int sockfd, const struct sockaddr \*serv\_addr, socklen\_t addrlen);

1. Funciones de envío y recibo de datos

un socket es un descriptor de fichero, así que en teoría sería posible escribir con write() y leer con read(), pero hay funciones mucho más cómodas para hacer esto. Dependiendo si el socket que utilicemos es de tipo socket de flujo o socket de datagramas emplearemos unas funciones u otras:

* Para sockets de flujo: send() y recv().
* Para sockets de datagramas: sendto() y recvfrom().

int send (int s, const void \*msg, size\_t len, int flags);

int sendto (int s, const void \*msg, size\_t len, int flags, const struct sockaddr \*to, socklen\_t tolen);

int recv (int s, void \*buf, size\_t len, int flags);

int recvfrom (int s, void \*buf, size\_t len, int flags, struct sockaddr \*from, socklen\_t \*fromlen);

1. Función close()

Esta función cierra el descriptor de fichero del socket, liberando el socket y denegando cualquier envío o recepción a través de este. Para un mejor control de este, puede utilizarse la función shutdown().

int shutdown (int s, int how);

## **Programa realizado**

### **Programa del cliente**

A continuación, se presenta el código correspondiente al cliente, se encuentra comentada línea por línea, de esta forma no será necesario explicar más.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Imagen 9. Programa del cliente.

### **Programa del servidor**

A continuación, se presenta el código correspondiente al servidor, la cual se encuentra comentada línea por línea, de esta forma no será necesario explicar más.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## **Pruebas de funcionamiento**

Primeramente, tenemos que correr el servidor.

Texto

Descripción generada automáticamente

Imagen 10. Corriendo el servidor.

Posteriormente ya podemos ejecutar el cliente un máximo de 5 veces, pues son los usuarios permitidos.

Texto

Descripción generada automáticamente

Imagen 11. Corriendo el cliente.

Como podemos ver el cliente nos pide un nombre de usuario, al ingresarlo y si estamos dentro del rango de usuarios permitidos ya podremos comenzar a chatear.

Texto

Descripción generada automáticamente

Imagen 12. Conectando al chat.

Podemos ver que el servidor va mostrando las interacciones que los usuarios hacen.

Texto

Descripción generada automáticamente con confianza media

Imagen 13. Servidor.

Cuando otro usuario se conecta se le avisa a los demás usuarios, de igual forma el servidor registra estas acciones como se puede ver en las siguientes imágenes.

Texto

Descripción generada automáticamente

Imagen 14. Conexión de nuevo usuario.

Texto

Descripción generada automáticamente

Imagen 15. Aviso a otros usuarios.

Texto

Descripción generada automáticamente

Imagen 16. Servidor y sus acciones.

De esta forma podemos empezar a chatear.

Texto

Descripción generada automáticamente

Imagen 17. Vista del usuario Brandon.

Texto

Descripción generada automáticamente

Imagen 18. Vista del usuario Eduardo.

Cuando llegamos al limite soportado de usuarios no se podrá conectar ese usuario y el servidor arrojará un mensaje como podemos ver en las siguientes capturas.

Texto

Descripción generada automáticamente

Imagen 19. Error en el usuario al alcanzar usuarios máximos.

Texto

Descripción generada automáticamente

Imagen 20. Mensaje del servidor al alcanzar los usuarios máximos.

Finalmente, se mostrará a los usuarios y en el servidor cuando un usuario se salga del chat o finalice su hilo de ejecución, es decir, cierre el programa.

Texto

Descripción generada automáticamente

Imagen 21. Salida del usuario Brandon.

Texto

Descripción generada automáticamente

Imagen 22. Aviso de desconexión de un usuario.

Texto

Descripción generada automáticamente

Imagen 23. Mensajes del servidor.

De esta forma, si se desea, se pueden unir más usuarios.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Imagen 24. Conexión de nuevo usuario.

Texto

Descripción generada automáticamente

Imagen 25. Aviso a otros usuarios.

Texto

Descripción generada automáticamente

Imagen 26. mensajes del servidor

## **Instrucciones de ejecución**

Es importante ejecutar primeramente el servidor, en la siguiente imagen se muestra cómo hacerlo:



Imagen 27. Compilación y ejecución del servidor.

Siendo la primera línea la compilación y la segunda la ejecución.

Una vez compilado y ejecutado el servidor podemos compilar y ejecutar el cliente de la siguiente forma:



Imagen 28. Compilación y ejecución del cliente.

Siendo la primera línea la compilación y la segunda la ejecución.

Es importante que no se finalice el servidor antes de finalizar los procesos del cliente, pues podría ocurrir un error en los clientes al cerrarse la conexión con el servidor.

# **Conclusiones**

## **Fischer Salazar César Eduardo**

Durante el transcurso del curso se nos ha proporcionado información que nos servirían de cimientos que culminarían en la realización de este proyecto, dado que en prácticas anteriores hemos estado realizando la implementación de un socket crudo mediante su programación en lenguaje C así como el uso de manuales que nos brinda el sistema operativo de LINUX en cual se ha utilizado para observar el funcionamiento de estos.

El empleo practico de un socket y sus diferentes configuraciones me permitido observar las diferentes aplicaciones que estos pueden tener en ámbitos relacionados a las redes de computadora, pues la teoría puede llegar a abrumar un poco ya que es necesario tener identificados los manuales requeridos, por lo ya mencionado fue necesario investigar en fuentes secundarias sobre manuales dentro de LINUX que nos permitieran abrir un socket para establecer la conexión para llegar a un funcionamiento correcto.

Puedo concluir nuevamente que la realización de estos programas me ha parecido bastante interesante ya que estos me han permitido entender el cómo la implementación de un socket nos puede permitir obtener información sobre nuestra computadora, el capturar una trama y hasta el cómo poder tener una conexión bidireccional en vivo.

## **López García José Eduardo**

A lo largo del desarrollo de las prácticas relacionadas con esta materia, hemos aterrizado en los entendimientos claros acerca del manual de Linux, el uso de los sockets dentro de la programación en este sistema operativo, y diferentes comandos que tienen relación con estos elementos; con la teoría quedó bien entendido, y con las prácticas se ha visualizado su funcionamiento.

El empleo de los sockets, no solamente para el proyecto sino también para otros sistemas existentes relacionados con las redes, tiene una vital importancia dentro de esta rama, ya que ha sido posible comprender el funcionamiento de un chat en vivo, ya sea unidireccional o bidireccional. Claro, fue necesario buscar más allá de los recursos (las anteriores prácticas) con los que contábamos, para que su desarrollo fuera óptimo y pudiera tener un funcionamiento adecuado, dado que el empleo del lenguaje C para este tipo de sistemas es muy robusto y extenso, era un poco predecible que hubiera pequeños problemas dentro del desarrollo, pero se lograron solucionar.

En general, fue bastante satisfactorio poder realizar una sala de chat como este, así se pudo recalcar y comprobar para qué pueden ser empleados los sockets en la programación en C y con la asistencia del manual de Linux, y así poder pensar en las diferentes posibilidades que esto puede generar en su área; dentro de lo que cabe, el haber desarrollado el proyecto nos ha dado una apertura bastante amplia acerca del funcionamiento de un sistema de chats en vivo, como lo hemos estado utilizando en años recientes.

## **Meza Vargas Brandon David**

Los sockets, como bien hemos revisado a lo largo del semestre mediante las prácticas realizadas son muy importantes en las redes, con estos podemos crear desde un simple chat unidireccional o uno bidireccional como el realizado en este proyecto final. En el desarrollo del presente proyecto fue necesario investigar un poco más sobre los sockets y su uso en lenguaje c, lenguaje bajo el cual fue desarrollado este proyecto, llegando así a la conclusión que los sockets utilizados para este proyecto fue la mejor decisión ya que estos permitieron de forma correcta el chat bidireccional entre clientes conectados a un servidor el cual gestionaba las operaciones que los usuarios conectados a él hacían.

Es increíble lo que podemos hacer con los sockets con lenguaje c, es cierto que hay lenguajes que facilitan mucho el uso de sockets y no es complicado hacer lo realizado en este proyecto, pero hacerlo en lenguaje c trabajando con sockets como se hace en este lenguaje nos permitió obtener mucho conocimiento y experiencia en la programación de estos, siempre es importante saber como funcionan las cosas desde cero.

Es destacable mencionar que durante el desarrollo del proyecto se presentaron varias problemáticas, las cuales fueron en su mayoría resueltas con manuales proporcionados por Linux, tales como socket, 7 ip, y los manuales de las funciones usadas en el proyecto.

Un proyecto donde se vio reflejado lo visto a lo largo de las prácticas aterrizándolo a un entorno práctica y entretenido como lo fue este chat room.

# **Referencias**

* Speed Check. (Sin fecha). *Socket*. <https://www.speedcheck.org/es/wiki/socket/>
* Paszniuk, R. (2013). *Sockets en C (Parte I) – Linux*. <https://www.programacion.com.py/noticias/sockets-en-c-parte-i-linux>