



UNIVERSIDAD CATÓLICA DE LA SANTÍSIMA CONCEPCIÓN

IN1082C

REDES DE COMPUTADORES

PROFESOR YASMANY PRIETO

Informe tarea 2

Autores:

Kevin Campos Venegas
Derqui Sanhueza Balboa
Maritxiu Castro Pulgar

Concepción, Junio de 2024

Índice

1. Implementación de sockets	2
1.1. En que consiste	2
1.2. Utilidad y motivación para desarrollarla	2
1.3. Herramientas utilizadas en el desarrollo y operación de la aplicación .	2
1.4. Partes principales del código explicadas (funciones, estructuras de da- tos, etc.)	3
1.5. Justificar selección de socket UDP o TCP	6
1.6. Capturas de pantalla de operación del servidor y el cliente	7
1.7. Capturas de pantalla de paquetes en Wireshark e identificar a que procesos de la comunicación pertenecen	12
1.8. Alcance de la aplicación y posibles mejoras	18
1.9. Opcionalmente un resumen sobre lo que se aprendió de la experiencia o si se experimentó alguna dificultad específica.	18

1. Implementación de sockets

1.1. En que consiste

La implementación de sockets TCP presentada consiste en una aplicación cliente-servidor que simula un sistema bancario. Al conectarse al servidor, el cliente puede crear una cuenta bancaria proporcionando sus datos personales (RUT, clave y saldo inicial). Una vez registrado y autenticado, el usuario puede realizar diversas operaciones básicas, tales como consultar saldo, transferir fondos, cerrar sesión y salir de la aplicación (lo cual termina la comunicación con el servidor).

1.2. Utilidad y motivación para desarrollarla

La utilidad de esta aplicación es permitir a los clientes bancarios realizar operaciones básicas similar a una sucursal virtual bancaria, tales como transacciones monetarias. Al ser una aplicación ejecutada mediante consola proporciona al cliente rapidez además de la seguridad proporcionada según el protocolo TCP.

La motivación para realizar esta aplicación fue el aprendizaje y entendimiento del funcionamiento de los sockets TCP en operaciones de intercambio de información entre cliente y servidor. A través del desarrollo del sistema bancario, se exploraron las técnicas de configuración y gestión de conexiones TCP. Además, el uso de la herramienta Wireshark permitió visualizar y analizar el flujo de comunicación entre el cliente y el servidor, proporcionando una comprensión práctica y detallada de la transmisión y recepción de datos en una red.

1.3. Herramientas utilizadas en el desarrollo y operación de la aplicación

La implementación de estos sockets TCP se desarrolla en el lenguaje C en una máquina virtual de sistema operativo Linux.

La implementación de estos sockets TCP se desarrolla en un entorno Linux mediante el lenguaje de programación C, haciendo uso de ciertas bibliotecas para la implementación de los sockets que requieren un sistema operativo basado en Unix ya que algunos headers no están disponibles en sistema operativo Windows.

Las algunas de las bibliotecas utilizadas son:

- sys/socket.h
- netinet/in.h
- arpa/inet.h
- netdb.h

1.4. Partes principales del código explicadas (funciones, estructuras de datos, etc.)

- En el servidor:

```
typedef struct {
    char rut[12];
    char clave[20];
    float saldo;
} Usuario;
```

Figura 1: Estructura

El servidor declara una estructura base para los Usuarios con los parámetros rut, clave y saldo

```

/*****************/
/* funciones auxiliares */
/*****************/
int buscar_usuario(char *rut)
{
    for (int i = 0; i < num_usuarios; i++) {
        if (strcmp(usuarios[i].rut, rut) == 0) {
            return i;
        }
    }
    return -1;
}

void registrar_usuario(char *rut, char *clave, float saldo_inicial)
{
    strcpy(usuarios[num_usuarios].rut, rut);
    strcpy(usuarios[num_usuarios].clave, clave);
    usuarios[num_usuarios].saldo = saldo_inicial;
    num_usuarios++;
}

int autenticar_usuario(char *rut, char *clave)
{
    int index = buscar_usuario(rut);
    if (index != -1 && strcmp(usuarios[index].clave, clave) == 0) {
        return index;
    }
    return -1;
}

```

Figura 2: Funciones básicas servidor

Definimos las funciones básicas del servidor `buscar_usuario()` se encarga de retornar el índice del usuario registrado consultándolo por el rut, `registrar_usuario()` se encarga de en base a los parámetros de entrada rut, clave, saldo_inicial registrar un nuevo usuario en el array de Usuarios (usuarios es una STRUCT), adicionalmente incrementamos la variable num_usuarios para llevar un registro del usuario que se conecta, luego tenemos la función `autenticar_usuario()` que se encarga de con un rut y una clave, verificar que el Usuario registrado corresponde a la clave ingresada, con esto le damos acceso a las funciones como depositar a otros usuarios.

```

void manejar_mensaje(char *mensaje_entrada, char *mensaje_salida)
{
    char comando[20], rut[12], clave[20];
    float monto;
    int usuario_index;

    sscanf(mensaje_entrada, "%s", comando);

    if (strcmp(comando, "REGISTRAR") == 0) {
        sscanf(mensaje_entrada, "%s %s %f", comando, rut, clave);
        if (buscar_usuario(rut) == -1) {
            registrar_usuario(rut, clave, monto);
            sprintf(mensaje_salida, "Usuario %s registrado con saldo inicial %.2f", rut, monto);
        } else {
            sprintf(mensaje_salida, "Error: Usuario %s ya existe", rut);
        }
    } else if (strcmp(comando, "AUTENTICAR") == 0) {
        sscanf(mensaje_entrada, "%s %s %s", comando, rut, clave);
        usuario_index = autenticar_usuario(rut, clave);
        if (usuario_index != -1) {
            sprintf(mensaje_salida, "Usuario %s autenticado correctamente", rut);
        } else {
            sprintf(mensaje_salida, "Error: Autenticación fallida para %s", rut);
        }
    } else if (strcmp(comando, "CONSULTAR_SALDO") == 0) {
        sscanf(mensaje_entrada, "%s %s", comando, rut, clave);
        usuario_index = autenticar_usuario(rut, clave);
        if (usuario_index != -1) {
            sprintf(mensaje_salida, "Saldo de %s %.2f", rut, usuarios[usuario_index].saldo);
        } else {
            sprintf(mensaje_salida, "Error: Autenticación fallida para %s", rut);
        }
    } else if (strcmp(comando, "TRANSFERIR") == 0) {
        char rut_destino[12];
        sscanf(mensaje_entrada, "%s %s %s %s %f", comando, rut, clave, rut_destino, &monto);
        usuario_index = autenticar_usuario(rut, clave);
        int destino_index = buscar_usuario(rut_destino);
        if (usuario_index >= 0 && destino_index != -1 && usuarios[usuario_index].saldo >= monto) {
            usuarios[destino_index].saldo += monto;
            usuarios[usuario_index].saldo -= monto;
            sprintf(mensaje_salida, "Transferencia de %.2f de %s a %s realizada con éxito", monto, rut, rut_destino);
        } else {
            sprintf(mensaje_salida, "Error en la transferencia: Verifique datos y saldo");
        }
    } else if (strcmp(comando, "terminar();") == 0) {
        sprintf(mensaje_salida, "Terminando conexión con cliente");
    } else {
        sprintf(mensaje_salida, "Comando no reconocido");
    }
}

```

Figura 3: Recepción del mensaje servidor

Es la función principal del servidor, maneja las solicitudes en formato string para darles un sentido y lógica a procesar tenemos los siguientes comandos:

- **REGISTRAR** -secciona el string para recibir el rut, clave y monto
- **AUTENTICAR** -envia a la sección para recibir usuario y contraseña
- **CONSULTAR_SALDO** -si esta autenticado puede consultar la variable saldo de su cuenta autenticada
- **TRANSFERIR** -secciona un string “TRANSFERIR rut clave rut_destinatario, monto” para realizar una transferencia a otra persona en el array de Usuarios
- **terminar();** -envia la finalización de la conexión

```
//Crear hilo para manejar el cliente
if (pthread_create(&hilo_cliente, NULL, manejar_cliente, &descriptor_socket_cliente) != 0)
{
    printf("ERROR: No se pudo crear el hilo para manejar el cliente\n");
    close(descriptor_socket_cliente);
    continue;
}

// Desatachar hilo para que no se quede en estado 'zombie'
pthread_detach(hilo_cliente);
```

Figura 4: Manejo de cada cliente en un hilo

como tenemos las funciones de los clientes en una función principal, podemos correr estas funciones como hilos de ejecución, como no podemos tener hilos infinitos tenemos un máximo de clientes que podemos procesar simultáneamente

- **En el cliente:** el código base del cliente es mas sencillo, se limita a recibir la

```
/*
* Ejecución: ./cliente [IP destino] [puerto]
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <stropts.h>
#include <stropts.h>

int main(int argc, char *argv[])
{
    struct sockaddr_in socket_destino;
    char respuesta[MAX_TAM_MENSAJE];
    int recibidos, enviados; // bytes recibidos y enviados

    if [argc != 3]-
        // Creamos el socket del cliente
        descriptor_socket_cliente = socket(AF_INET, SOCK_STREAM, 0);
    if (descriptor_socket_cliente == -1)-
        // Se prepara la dirección de la máquina servidora
        socket_destino.sin_family = AF_INET;
        socket_destino.sin_addr.s.addr = inet_addr(argv[1]);
        socket_destino.sin_port = htons(atoi(argv[2]));

        // Establece una conexión con el servidor
        if (connect(descriptor_socket_cliente, (struct sockaddr *)&socket_destino, sizeof(socket_destino)) == -1) {
            signal(SIGINT, &catch);
            while (1)-
                // Se cierra la conexión (socket)
                printf("\ncliente termina.\n");
                close(descriptor_socket_cliente);
                exit(EXIT_SUCCESS);
        }
}
```

Figura 5: Código base del cliente

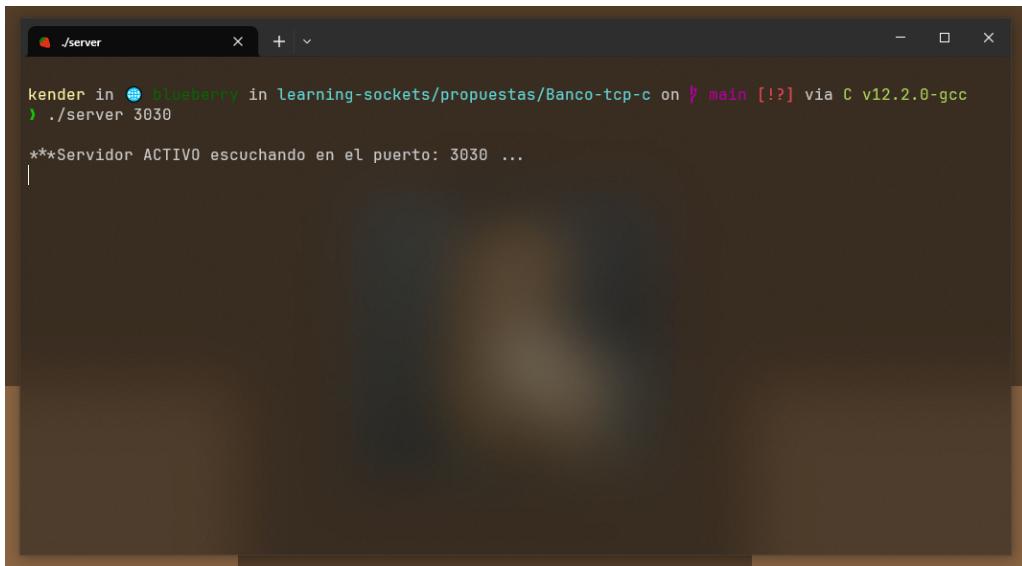
información del servidor e iniciar la conexión, valida los argumentos de entrada que sean una IP y un PUERTO, además de tener un ciclo while que tiene el menú de opciones y un switch case para enviar los strings con los comandos que se le piden al usuario desde el menú.

1.5. Justificar selección de socket UDP o TCP

Ya que la aplicación trabajara con datos sensibles y la fidelidad de los datos es fundamental, decidimos por usar el protocolo TCP ya que si bien no es el mas rápido, este permitirá tener una conexión cliente/servidor fidedigna donde priorizaremos no tener ambigüedades sobre las transferencias de montos ni inicios de sesión de clientes que no son ellos. sacrificamos velocidad por seguridad y consistencia.

1.6. Capturas de pantalla de operación del servidor y el cliente

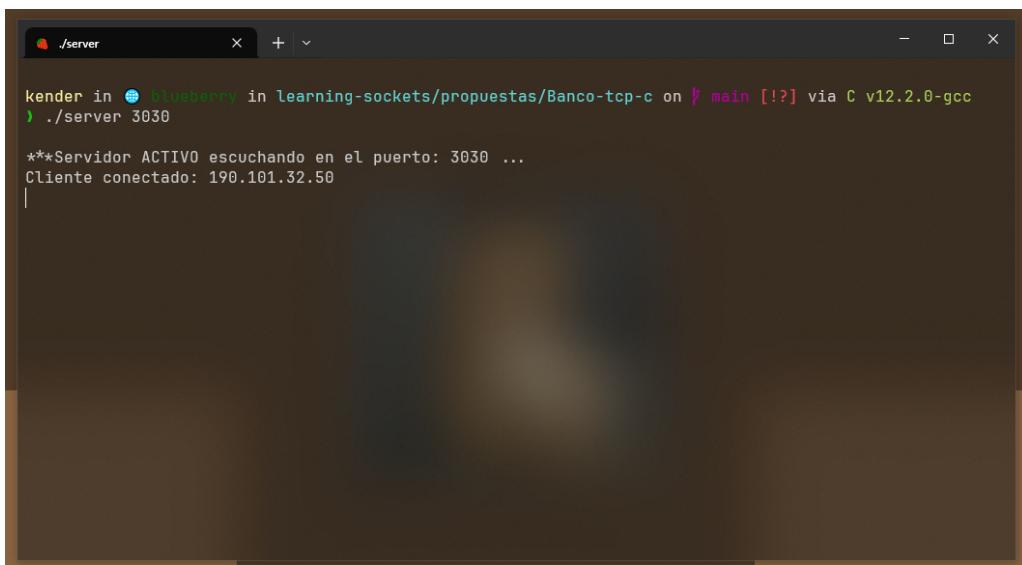
1. Establecimiento de la conexión



```
./server
kender in ● bluberry in learning-sockets/propuestas/Banco-tcp-c on ↵ main [!?] via C v12.2.0-gcc
> ./server 3030

***Servidor ACTIVO escuchando en el puerto: 3030 ...
```

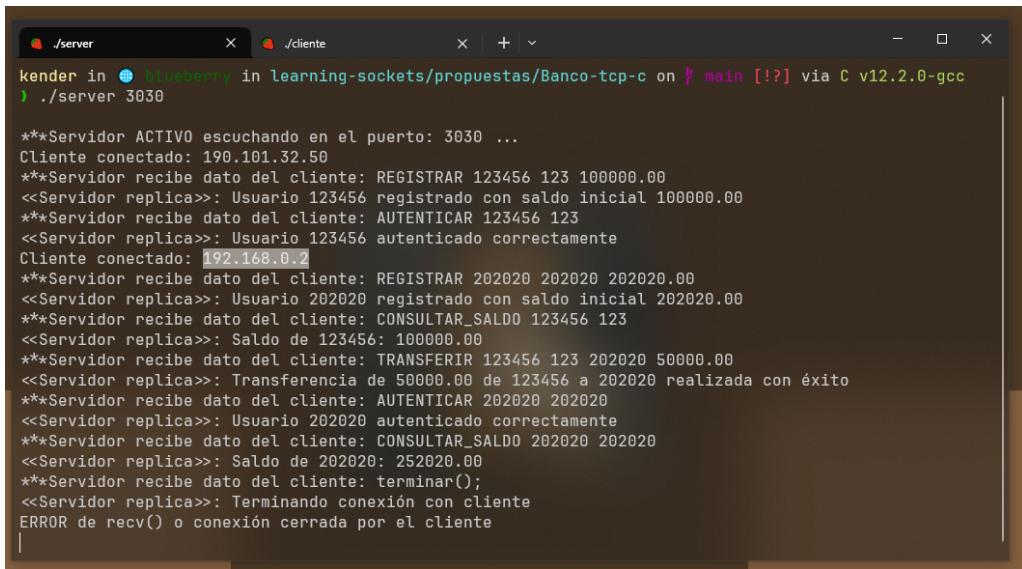
Figura 6: Se inicia servidor, servidor escuchando



```
./server
kender in ● bluberry in learning-sockets/propuestas/Banco-tcp-c on ↵ main [!?] via C v12.2.0-gcc
> ./server 3030

***Servidor ACTIVO escuchando en el puerto: 3030 ...
Cliente conectado: 190.101.32.50
```

Figura 7: Se conecta un cliente al servidor



A screenshot of a terminal window titled 'server' showing the output of a C program. The program is a simple TCP server for a banking application. It logs messages indicating client connections, registrations, logins, and transactions. The server is listening on port 3030 and has received a connection from a client at 190.101.32.50. The client performs a registration, logs in, checks its balance, transfers 50000.00 to another account, and then disconnects.

```
kender in blueberry in learning-sockets/propuestas/Banco-tcp-c on main [!?] via C v12.2.0-gcc
> ./server 3030

***Servidor ACTIVO escuchando en el puerto: 3030 ...
Cliente conectado: 190.101.32.50
**Servidor recibe dato del cliente: REGISTRAR 123456 123 100000.00
<<Servidor replica>>: Usuario 123456 registrado con saldo inicial 100000.00
**Servidor recibe dato del cliente: AUTENTICAR 123456 123
<<Servidor replica>>: Usuario 123456 autenticado correctamente
Cliente conectado: 192.168.0.2
***Servidor recibe dato del cliente: REGISTRAR 202020 202020 202020.00
<<Servidor replica>>: Usuario 202020 registrado con saldo inicial 202020.00
***Servidor recibe dato del cliente: CONSULTAR_SALDO 123456 123
<<Servidor replica>>: Saldo de 123456: 100000.00
**Servidor recibe dato del cliente: TRANSFERIR 123456 123 202020 50000.00
<<Servidor replica>>: Transferencia de 50000.00 de 123456 a 202020 realizada con éxito
**Servidor recibe dato del cliente: AUTENTICAR 202020 202020
<<Servidor replica>>: Usuario 202020 autenticado correctamente
***Servidor recibe dato del cliente: CONSULTAR_SALDO 202020 202020
<<Servidor replica>>: Saldo de 202020: 252020.00
**Servidor recibe dato del cliente: terminar();
<<Servidor replica>>: Terminando conexión con cliente
ERROR de recv() o conexión cerrada por el cliente
```

Figura 8: Servidor recibe conexión de cliente

2. Intercambio de datos

The screenshot shows a terminal window with two tabs open. The left tab is titled './server' and the right tab is titled './cliente'. Both tabs are running on a Mac OS X system, as indicated by the window icons.

The ./cliente session is active and displays the following interaction:

```
kender in bluberry in Learning-Sockets/propuestas/Banco-tcp-c on main [!?] via C v12.2.0-gcc
> ./cliente 192.168.0.2 3030

Opciones:
1. Registrar usuario
2. Autenticar usuario
3. Consultar saldo
4. Transferir fondos
5. Cerrar sesión
6. Salir

Seleccione una opción: 1
Ingrese RUT: 202020
Ingrese clave: 202020
Ingrese saldo inicial: 202020
<<Servidor>>: Usuario 202020 registrado con saldo inicial 202020.00

Opciones:
1. Registrar usuario
2. Autenticar usuario
3. Consultar saldo
4. Transferir fondos
5. Cerrar sesión
6. Salir

Seleccione una opción: |
```

Figura 9: Se registra un cliente al servidor

The screenshot shows two terminal windows side-by-side. The left window, titled '/server', displays the command: 'kender in blueberry in learning-sockets/propuestas/Banco-tcp-c on main [!?] via C v12.2.0-gcc'. Below this, it shows the output of a client program: 'cliente 192.168.0.2 3030'. The right window, titled '/cliente', shows the client's menu options:

```
Opciones:  
1. Registrar usuario  
2. Autenticar usuario  
3. Consultar saldo  
4. Transferir fondos  
5. Cerrar sesión  
6. Salir  
Seleccione una opción: 1  
Ingrese RUT: 202020  
Ingrese clave: 202020  
Ingrese saldo inicial: 202020  
<<Servidor>>: Usuario 202020 registrado con saldo inicial 202020.00
```

After this, the client menu reappears:

```
Opciones:  
1. Registrar usuario  
2. Autenticar usuario  
3. Consultar saldo  
4. Transferir fondos  
5. Cerrar sesión  
6. Salir  
Seleccione una opción: |
```

Figura 10: Registro de otro cliente

This screenshot shows the same two terminal windows. The left window ('/server') shows the server confirming the user was authenticated correctly: '<<Servidor>>: Usuario 202020 autenticado correctamente'. The right window ('/cliente') shows the client menu again. A transaction is performed:

```
6. Salir  
Seleccione una opción: 2  
Ingrese RUT: 202020  
Ingrese clave: 202020  
<<Servidor>>: Usuario 202020 autenticado correctamente  
  
Opciones:  
1. Registrar usuario  
2. Autenticar usuario  
3. Consultar saldo  
4. Transferir fondos  
5. Cerrar sesión  
6. Salir  
Seleccione una opción: 3  
<<Servidor>>: Saldo de 202020: 252020.00  
  
Opciones:  
1. Registrar usuario  
2. Autenticar usuario  
3. Consultar saldo  
4. Transferir fondos  
5. Cerrar sesión  
6. Salir  
Seleccione una opción: |
```

Figura 11: Cuenta de usuario recibe el monto

3. Finalización de la conexión

The screenshot shows a terminal window titled "superos@superos-VirtualBox: ~/Escritorio/sockets_tcp". The window contains the following text:

```
Ingrese clave: 123
<<Servidor>>: Usuario 123456 autenticado correctamente

Opciones:
1. Registrar usuario
2. Autenticar usuario
3. Consultar saldo
4. Transferir fondos
5. Cerrar sesión
6. Salir

Seleccione una opción: 3
<<Servidor>>: Saldo de 123456: 100000.00

Opciones:
1. Registrar usuario
2. Autenticar usuario
3. Consultar saldo
4. Transferir fondos
5. Cerrar sesión
6. Salir

Seleccione una opción: 4
Ingrese RUT del destinatario: 202020
Ingrese monto a transferir: 50000
<<Servidor>>: Transferencia de 50000.00 de 123456 a 202020 realizada con éxito

Opciones:
1. Registrar usuario
2. Autenticar usuario
3. Consultar saldo
4. Transferir fondos
5. Cerrar sesión
6. Salir

Seleccione una opción: 6
Saliendo...
<<Servidor>>: Terminando conexión con cliente

Cliente termina.
superos@superos-VirtualBox:~/Escritorio/sockets_tcp$
```

Figura 12: Finalización de la conexión

1.7. Capturas de pantalla de paquetes en Wireshark e identificar a que procesos de la comunicación pertenecen

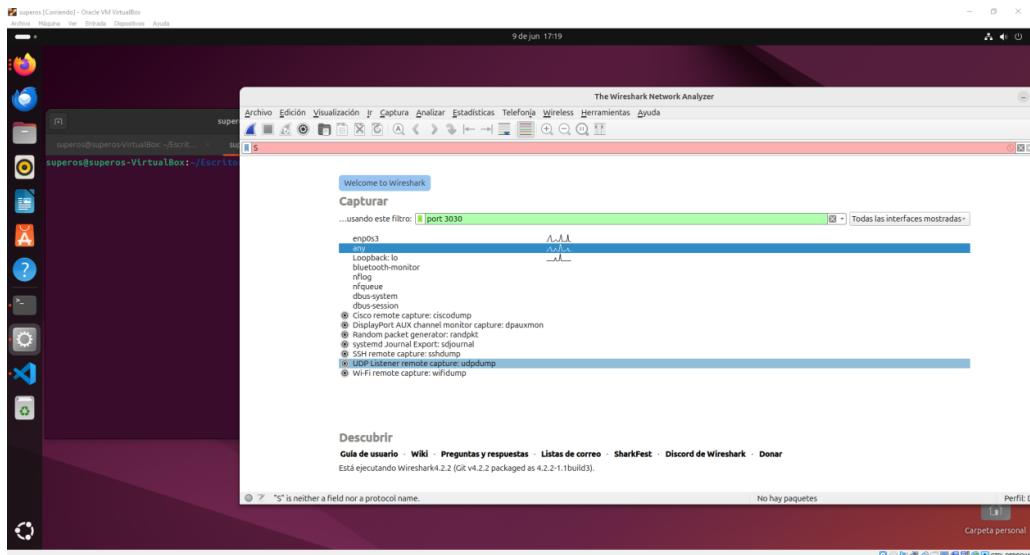


Figura 13: Filtro por puerto 3030 en Wireshark

1. Establecimiento de la conexión

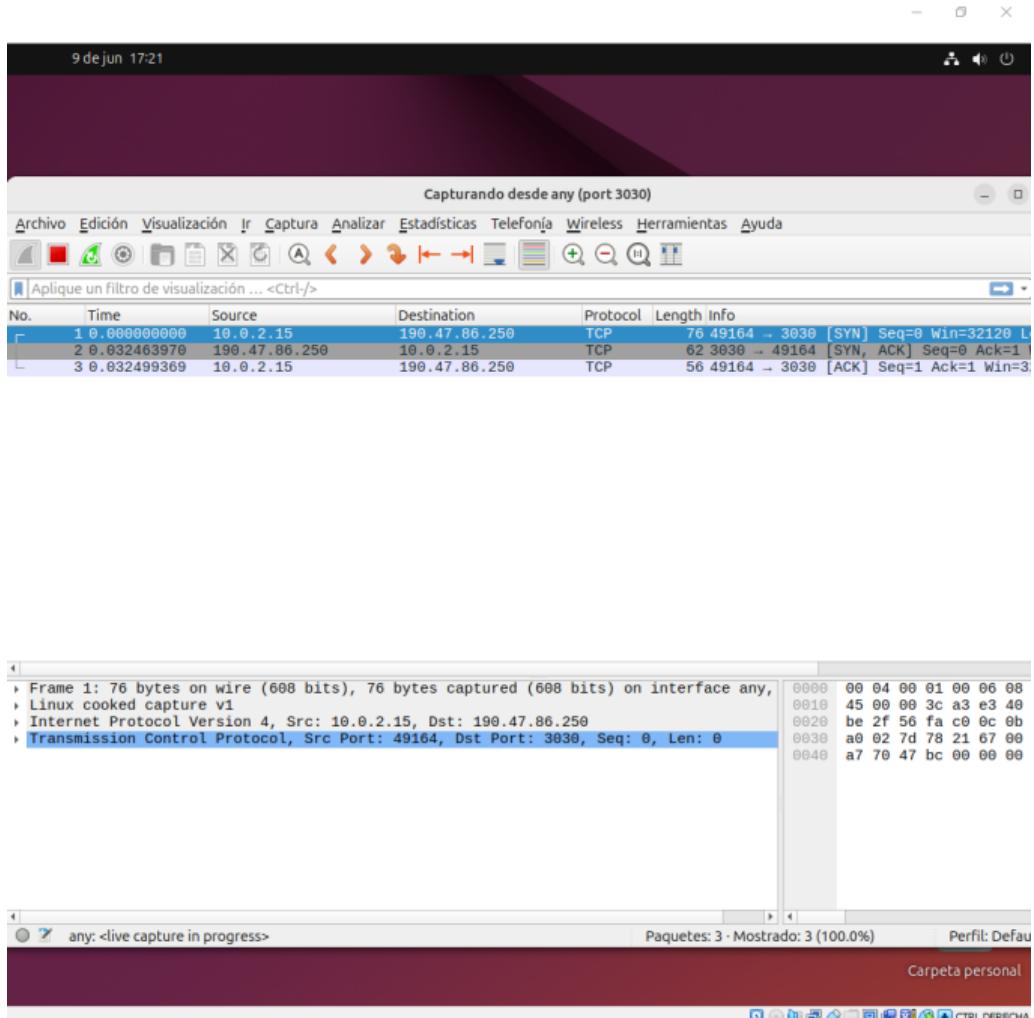


Figura 14: Establecimiento de la conexión exitosa

2. Intercambio de datos

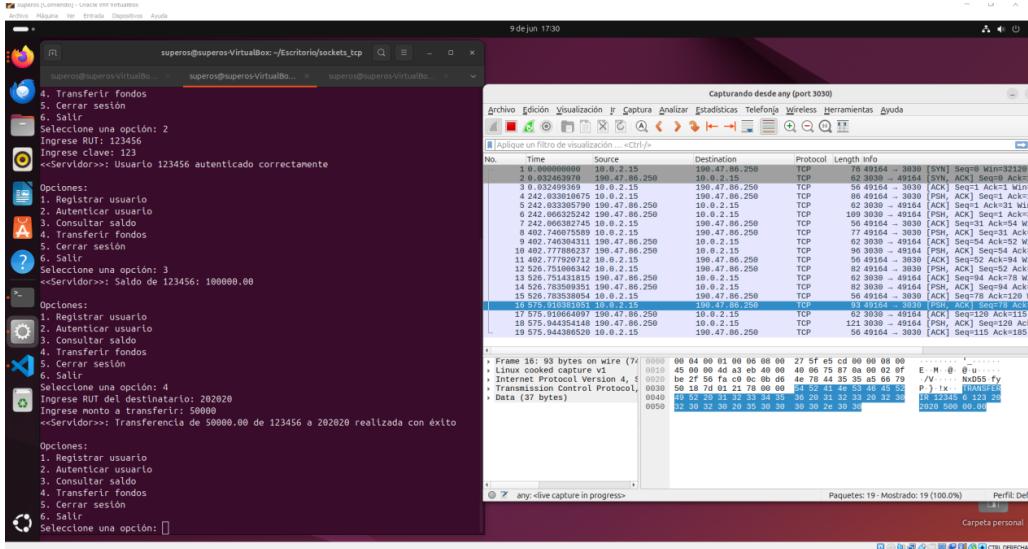


Figura 15: Se realiza una transferencia

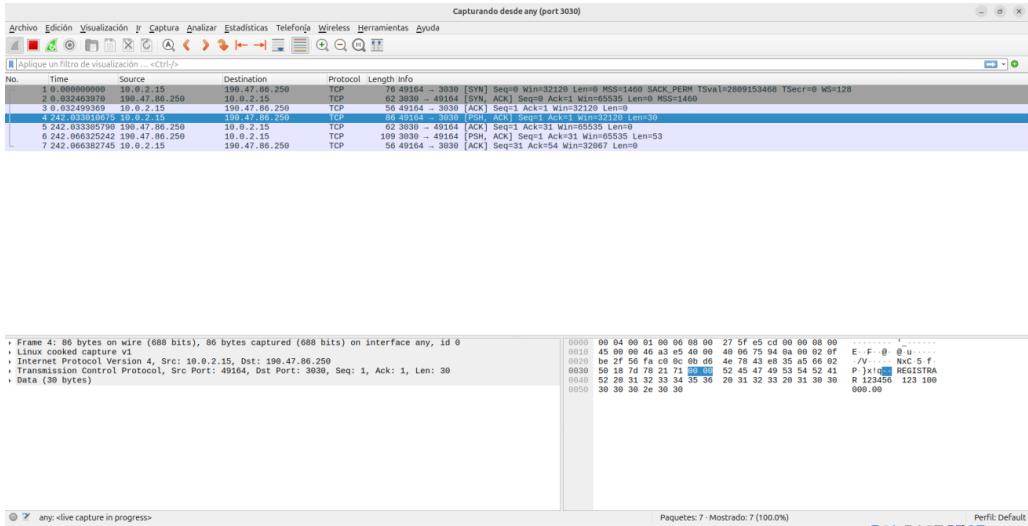


Figura 16: Paquete de registro de usuario

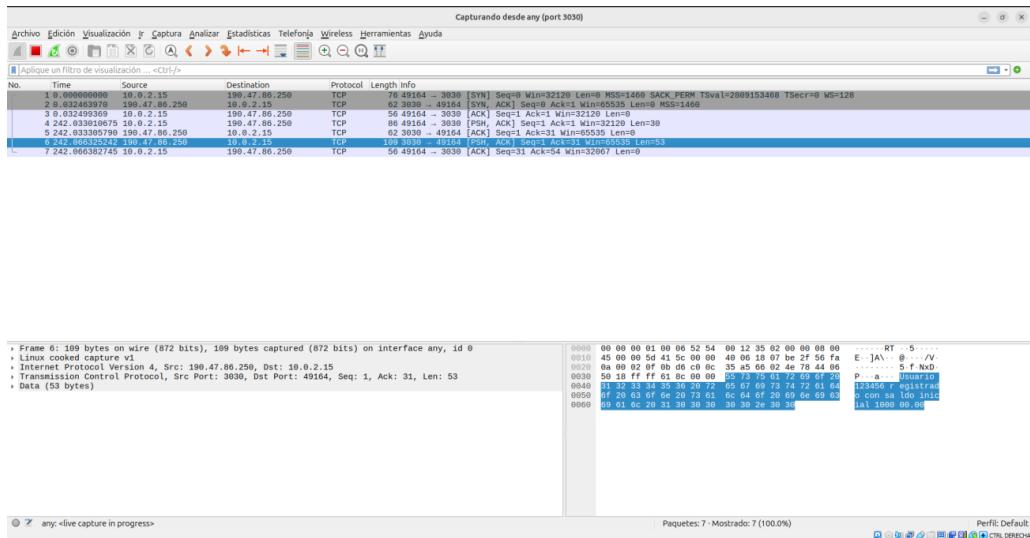


Figura 17: Retorno servidor

3. Finalización de la conexión

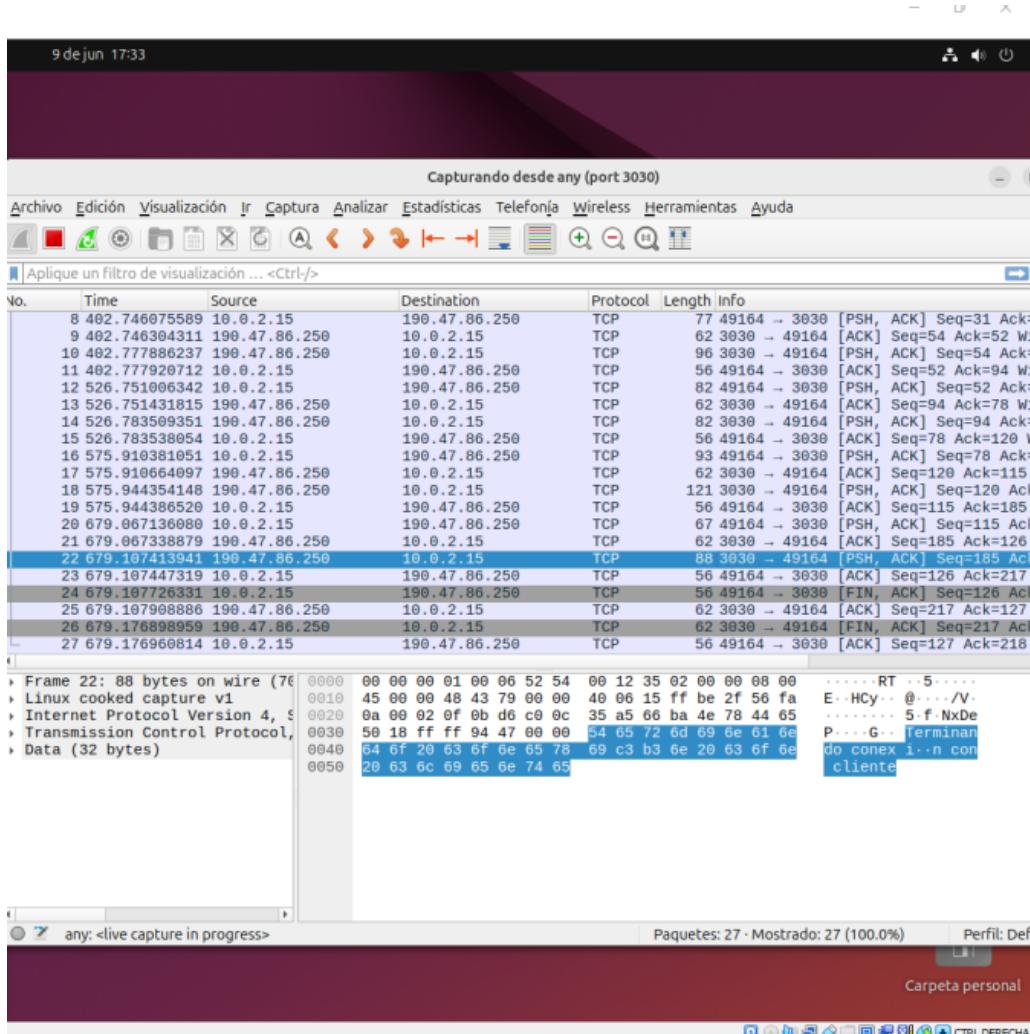


Figura 18: Finalización de la conexión

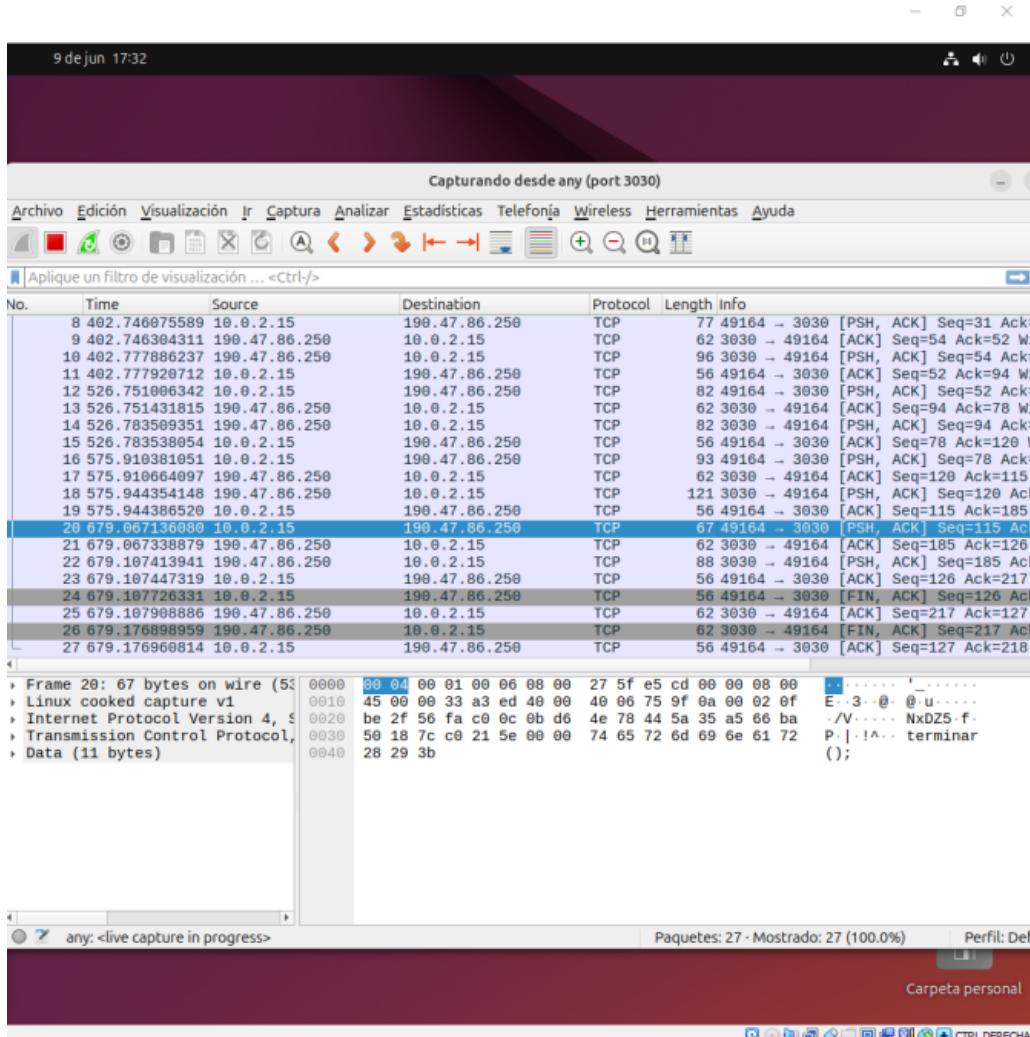


Figura 19: Cierre de conexión

En las imágenes podemos identificar los headers de [FIN] y [ACK] en las filas grises respectivos que indican que se finalizara la conexión entre cliente y servidor

1.8. Alcance de la aplicación y posibles mejoras

Dentro de las mejoras que se consideran pertinentes para esta aplicación es la implementación de interfaz de usuario mas agradable, agregar más funcionalidades como revisión de depósitos, simulación de depósitos a plazo, persistencia de datos, etc.

1.9. Opcionalmente un resumen sobre lo que se aprendió de la experiencia o si se experimentó alguna dificultad específica.

Para posibilitar que más de un cliente pudiese realizar operaciones de manera simultanea con el servidor, se implemento el uso de hilos, de esta forma, el servidor es capaz de sostener la conexión de dos o más clientes realizando operaciones de manera simultanea. Adicionalmente y al analizar la conexión a través de wireshark logramos identificar las distintas partes de cada paquete y cual es la importancia de cada uno de estos headers para que la comunicación sea segura o rápida.