PRÁCTICA 3 – CLIENTE SERVIDOR

FECHA: 11/10/21

NOMBRE DEL EQUIPO: EL SIUU TEAM

PARTICIPANTES: -FISCHER SALAZAR CÉSAR EDUARDO
-LÓPEZ GARCÍA JOSÉ EDUARDO
-MEZA VARGAS BRANDON DAVID

UNIDAD ACADÉMICA: REDES DE COMPUTADORAS

PANORÁMICA

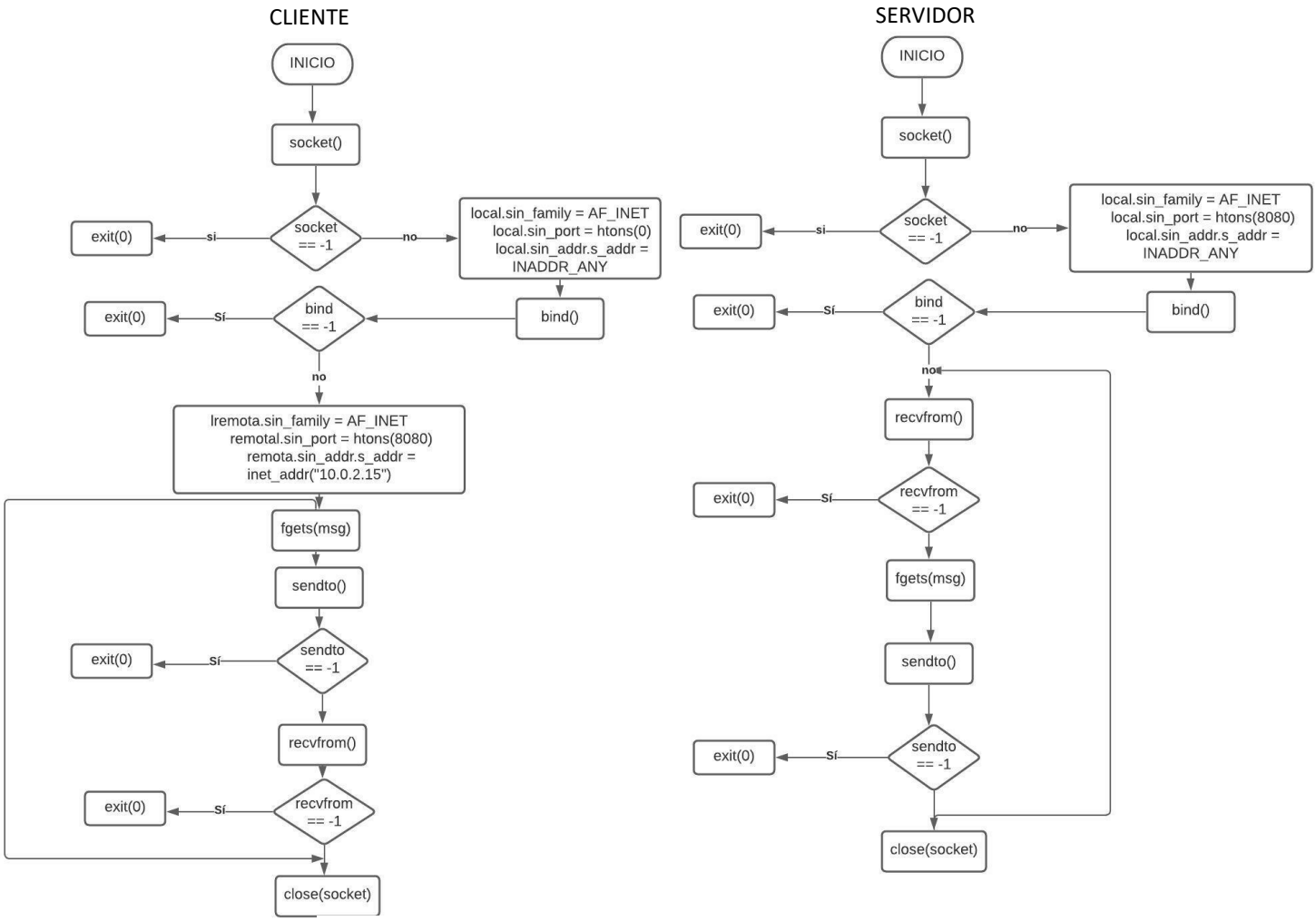

Figura 1. Diagrama de flujo del cliente y servidor

OBJETIVOS

OBJETIVO PRINCIPAL: PROGRAMAR SOCKETS, RELACIONAR LA FUNCIÓN BIND CON EL SOCKET, PROGRAMAR UN CLIENTE Y UN SERVIDOR QUE SEAN CAPACES DE ENTABLAR UNA COMUNICACIÓN ASINCRONA . APOYÁNDONOS CON EL USO DEL ANALIZADOR DE RED WIRESHARK CAPTURAR TRAMAS ENVIADAS POR EL CLIENTE Y EL SERVIDOR (CAPTURAR UNA TRAMA DE CADA UNO), DESCRIBIR LOS ENCABEZADOS DE CADA CAPA DEL MODELO TCP/IP

OBJETIVO SECUNDARIO. VERIFICAR ENVIÓ DE UN MENSAJE A TRAVÉS DE LA RED EN UN ESENARIO CLIENTE SERVIDOR.

ESCENARIO

LA FORMA MÁS BÁSICA DE CONSTRUIR UN SISTEMA DE COMUNICACIÓN ES IMPLEMENTANDO LA INTERFAZ DE PROGRAMACIÓN DE APLICACIONES DE SOCKET (API DE SOCKET - SOCKETS APLICACIÓN PROGRAMMING INTERFACE).

EN ESTA PRÁCTICA EL PARTICIPANTE DEBERÁ REALIZAR UN PROGRAMA QUE ENVÍE UN MENSAJE A TRAVÉS DE SU RED Y PODER SER CAPTURADO POR MEDIO DE UN SERVIDOR.

RECURSOS NECESARIOS PARA REALIZAR LA PRÁCTICA

- Compilador Linux y editor de Linux
- Manuales de linux
- Software wireshark

PARTE 1: DIAGRAMA DE FLUJO

- **INCLUYE DIAGRAMA DE FUJO DE CLIENTE Y SERVIDOR.**

CLIENTE:

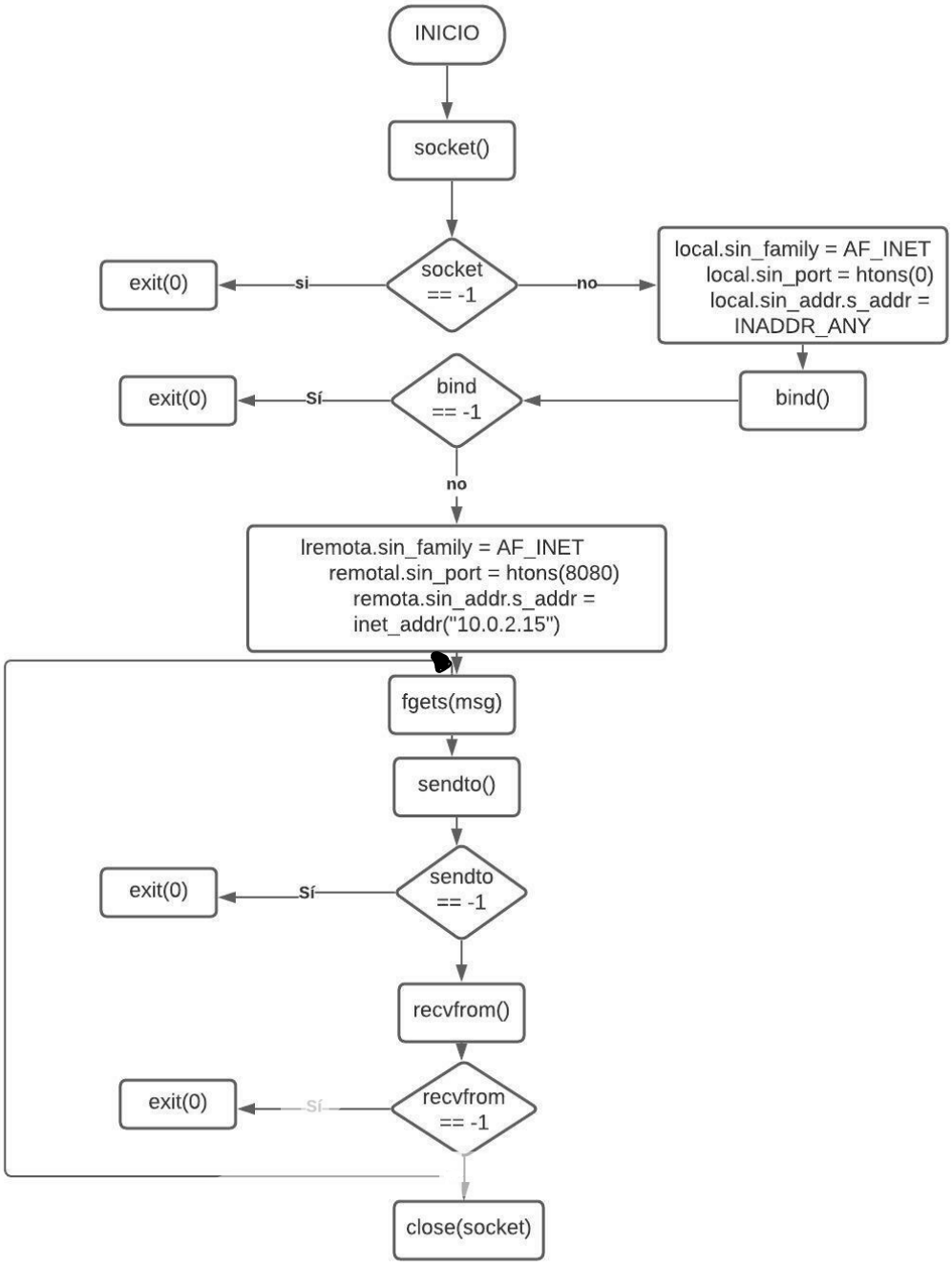

Figura 2. Diagrama de flujo del cliente-

SERVIDOR:

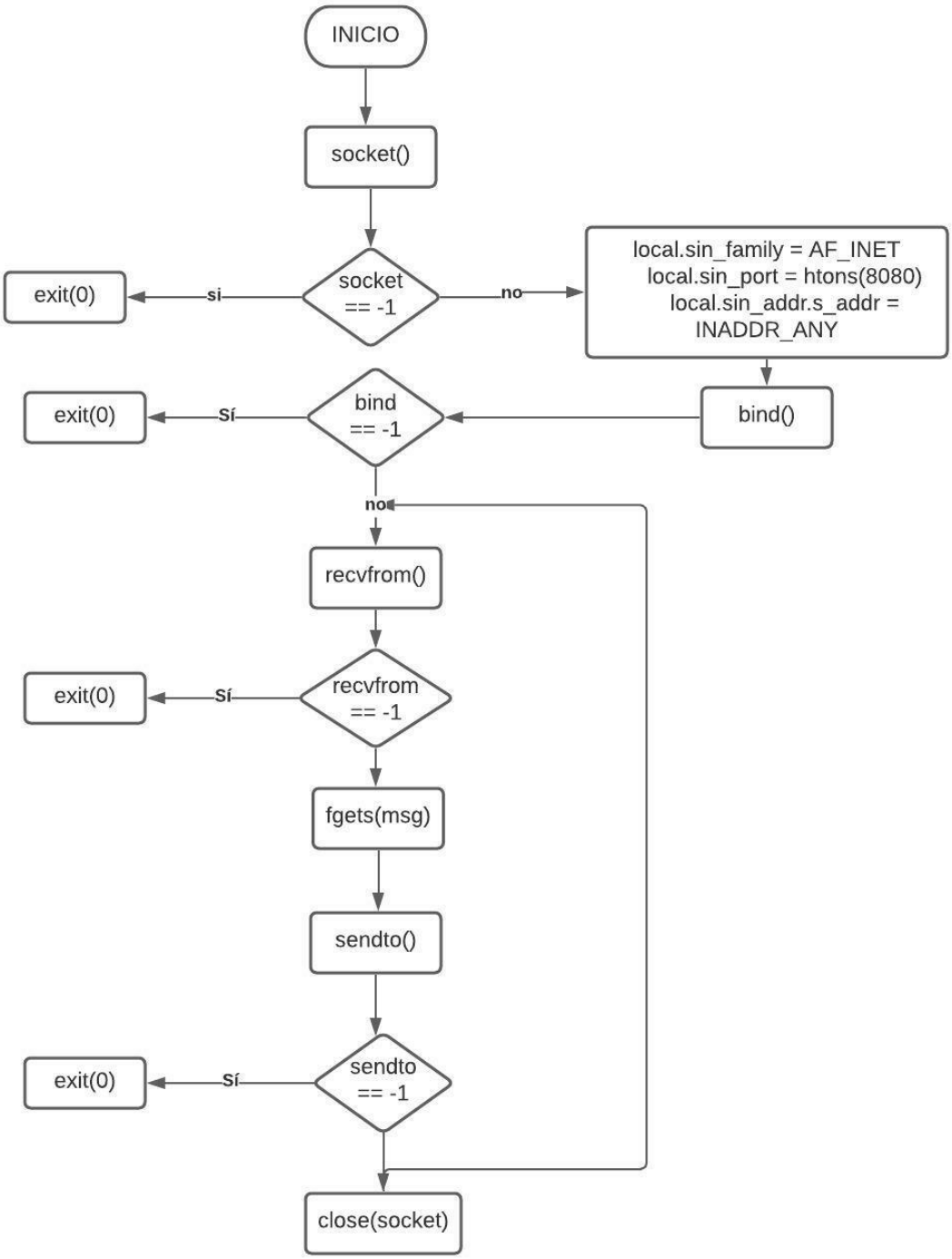

Figura 3. Diagrama de flujo del servidor

PARTE 2: CÓDIGOS, COMANDOS Y EJECUCIÓN Y EXPLICACIÓN.

2.1 INCLUIR CODIGOS EXPLICANDO LÍNEA POR LÍNEA CLIENTE Y SERVIDOR, CAMBIAR EL NOMBRE DE SUS VARIABLES Y ESTRUCTURAS DE FORMA PERSONAL. RECUERDEN QUE LAS MEJORAS QUE LE HAGAN AL PROGRAMA VISTO EN CLASE AUMENTA SU CALIFICACION.

CLIENTE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h> //L2 protocols
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio_ext.h>


int main(){
    //comValue: para sendto y rcvfrom, lengthRecv: medir tamaño de datos que me envian,
    //udp_socket será para el socket
    int udp_socket, bindValue, comValue, lengthRecv;
    struct sockaddr_in local, remota;//estructuras para el servidor y client
    unsigned char paqRec[512]; //Buffer para recibir información del servidor
    unsigned char msg[100];//arreglo para guardar los mensajes enviados


    printf("\n\n\t\t\t*****CHAT CLIENTE A SERVIDOR*****\n\n");

    /*
        socket nos permite abrir un socket, AF_INET es la familia y usamos SOCK_DGRAM para abrir un socket
        udp, el 0 es para el protocolo IP en la cabecera IP a envar o recibir
    */
    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

    //Aqui indicamos si el socket se abrio correctamente o no: -1 = Error, de otra forma se abre correctamente
    if(udp_socket == -1){
        perror("\nError al abrir el socket");
        exit(0);//salimos del programa
    }else{
        perror("\nExito al abrir el socket");//se abrio el socket
        local.sin_family = AF_INET; //address family: AF_INET
        local.sin_port = htons(0); //port in network byte order, la api de socket asigna el puerto
        local.sin_addr.s_addr = INADDR_ANY;//cualquiera al ser comunicacion local en la maquina
```

```c
/*
    Con bind enlazamos un nombre a un socket, comop parametros esta el socket, la direccion local en este
    caso servidor y el tamaño de la direccion
*/
bindValue = bind(udp_socket,(struct sockaddr *)&local,sizeof(local));

//Si bindValue es -1 no se pudo hacer el bind
if(bindValue == -1){
    perror("\nError en bind");
    exit(0);//salimos del programa
}else{
    perror("\nExito en el bind");//se hizo el bind
    remota.sin_family = AF_INET; //address family: AF_INET
    remota.sin_port = htons(8080); //port in network byte order, cambio puerto serv
    remota.sin_addr.s_addr = inet_addr("10.0.2.15"); //inet_addr: convertir cadena, cambia puerto servidor

    lengthRecv = sizeof(remota);//calculamos el comValueaño de la estrucutra remota

    //ciclo
    while(1){

        //Solicitando mensaje a enviar
        __fpurge(stdin);//Limpíamops buffer del teclado
        printf("\nIngresa un mensaje: ");
        fgets(msg, sizeof(msg),stdin);//esperamos a que se escriba un mensaje a enviar

        /*
            sendto la usamos para mandar mensajes a otro conector, aqui especificamos la direccion
            a donde transmitiremos el mensaje, en este caso al servidor
        */
        comValue = sendto(udp_socket, msg, strlen(msg)+1, 0, (struct sockaddr *)&remota, sizeof(remota));

        //Si devuelve -1 no se pudo enviar
        if(comValue == -1){
            perror("\nError al enviar");
            exit(0);//salimos
        }else
            perror("\nExito al enviar");

        /*
            para recibir usaremos la función recvfrom
            EL buffer se definde 512 bytes porque es la cantidad idonea en los sockets para recbir
            informacion
            Flags: 0 = manera bloqueante: para esperar a que me llegue algo
            &lengthRecv es el tamaño de remota
        */
        comValue = recvfrom(udp_socket, paqRec, sizeof(paqRec), 0, (struct sockaddr *)&remota, &lengthRecv);

        //Si hay error al recibir devuelve -1
        if(comValue == -1){
            perror("\nError al recibir");
            exit(0);//Salimos del programa
        }else
            printf("\nServidor: %s\n\n", paqRec);//Si no hay error, imprimimos el paquete recibido


    }
}

close(udp_socket);//cerramos el socket

return 0;
}
```

Figura 4. Programa cliente

SERVIDOR

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h> //L2 protocols
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/types.h>
#include <stdio_ext.h> //para __fpurge(stdin)

int main(){
    //comValue: para sendto y recvfrom, lengthRecv: medir tamaño de datos que me envian, udp_socket será para el socket
    int udp_socket, bindValue, comValue, lengthRecv;
    unsigned char msg[100]; //arreglo para guardar los mensajes enviados
    struct sockaddr_in servidor, cliente; //estructuras para el servidor y client
    unsigned char paqRec[512]; //Buffer para recibir información del cliente

    printf("\n\n\t\t\t*****CHAT SERVIDOR A CLIENTE*****\n\n");

    /*
        socket nos permite abrir un socket, AF_INET es la familia y usamos SOCK_DGRAM para abrir un socket
        udp, el 0 es para el protocolo IP en la cabecera IP a envar o recibir
    */
    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

    //Aqui indicamos si el socket se abrio correctamente o no: -1 = Error, de otra forma se abre correctamente
    if(udp_socket == -1){
        perror("\nError al abrir el socket");
        exit(0); //salimos del programa
    }else{
        perror("\nExito al abrir el socket"); //se abrio el socket
        servidor.sin_family = AF_INET; //address family: AF_INET
        servidor.sin_port = htons(8080); //port in network byte order, 8080: puerto para recibir info
        servidor.sin_addr.s_addr = INADDR_ANY; //cualquiera al ser comunicacion local en la maquina
/*
    Con bind enlazamos un nombre a un socket, comop parametros esta el socket, la direccion local en este
    caso servidor y el tamaño de la direccion
*/
bindValue = bind(udp_socket,(struct sockaddr *)&servidor,sizeof(servidor));

//Si bindValue es -1 no se pudo hacer el bind
if(bindValue == -1){
    perror("\nError en bind");
    exit(0); //salimos del programa
}else{
    perror("\nExito en el bind"); //se hizo el bind

    lengthRecv = sizeof(cliente); //calculamos el comValueaño de la estrucutra cliente

    //ciclo
    while(1){

        /*
            para recibir usaremos la función recvfrom
            EL buffer se definde 512 bytes porque es la cantidad idonea en los sockets para recbir informacion
            Flags: 0 = manera bloqueante: para esperar a que me llegue algo
            &lengthRecv es el tamaño de cliente
        */
        comValue = recvfrom(udp_socket, paqRec, sizeof(paqRec), 0, (struct sockaddr *)&cliente, &lengthRecv);

        //Si hay error al recibir devuelve -1
        if(comValue == -1){
            perror("\nError al recibir");
            exit(0); //Salimos del programa
        }else
            printf("\nCliente: %s\n\n", paqRec); //Si no hay error, imprimimos el paquete recibido

            //Solicitando mensaje a enviar al cliente
            __fpurge(stdin); //Limpíamops buffer del teclado
```

```c
            printf("\nIngresa un mensaje: ");
            fgets(msg, sizeof(msg),stdin); //esperamos a que se escriba un mensaje a enviar

            /*
                sendto la usamos para mandar mensajes a otro conector, aqui especificamos la direccion
                a donde transmitiremos el mensaje, en este caso al cliente
            */

            comValue = sendto(udp_socket, msg, strlen(msg)+1, 0, (struct sockaddr *)&cliente, sizeof(cliente));

            //Si devuelve -1 no se pudo enviar
            if(comValue == -1){
                perror("\nError al enviar");
                exit(0); //salimos
            }else
                perror("\nExito al enviar"); //Se envio correccomValueente
        }
    }

    close(udp_socket); //cerramos el socket

    return 0;
}
```

Figura 5. Programa servidor

2.2 INCLUYE LA CAPTURA DE PANTALLA AL MANDAR UN MENSAJE POR EL CLIENTE.

```
                        *****CHAT CLIENTE A SERVIDOR*****


Exito al abrir el socket: Success

Exito en el bind: Success

Ingresa un mensaje: Hola soy el cliente

Exito al enviar: Success
```

Figura 6. Mensaje desde el cliente

2.3 INCLUYE LA CAPTURA DE PANTALLA AL MANDAR UN MENSAJE POR EL SERVIDOR.

```
                    *****CHAT SERVIDOR A CLIENTE*****


Exito al abrir el socket: Success

Exito en el bind: Success

Cliente: Hola soy el cliente



Ingresa un mensaje: Hola yo soy el servidor

Exito al enviar: Success
```

Figura 7. Mensaje desde el servidor

2.4 INCLUYA LA CAPTURA DE PANTALLA DE LA TRAMA ENVIADA POR EL CLIENTE, CON AYUDA DEL PROGRAMA WIRE SHARK, Y EXPLIQUE LOS DATOS OBTENIDOS EN CADA CAPA DEL MODELO OSI.

La trama capturada del cliente fue la siguiente:

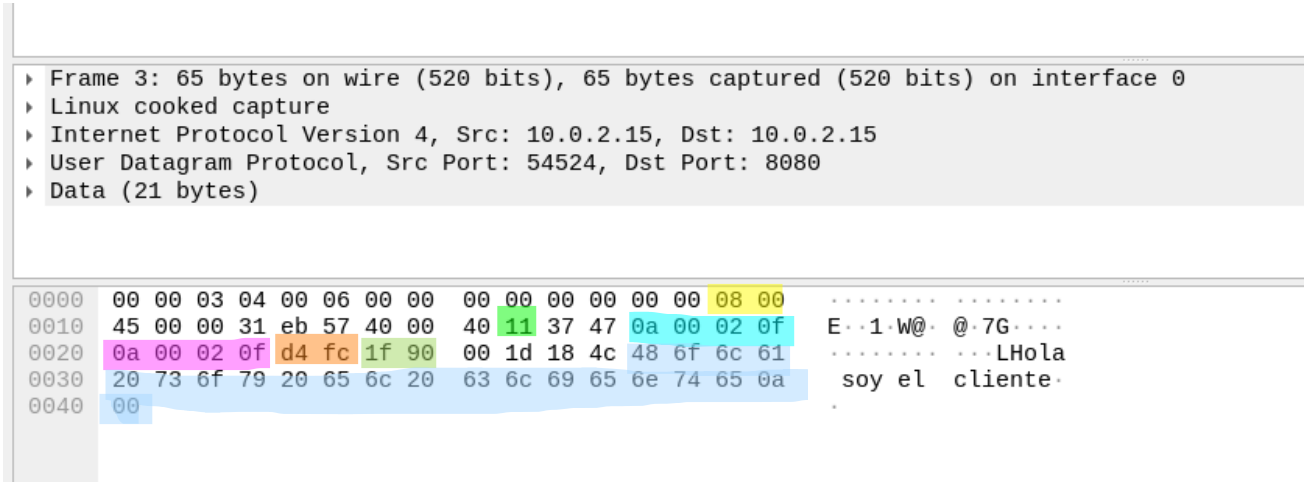

Figura 8. Trama del cliente

Protocolo IP

Puerto fuente

Protocolo UDP

Puerto destino

IP fuente

Mensaje

IP destino

2.5 INCLUYA LA CAPTURA DE PANTALLA DE LA TRAMA ENVIADA POR EL CLIENTE, CON AYUDA DEL PROGRAMA WIRE SHARK, Y EXPLIQUE LOS DATOS OBTENIDOS EN CADA CAPA DEL MODELO OSI.

```
▸ Frame 2: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15
▸ User Datagram Protocol, Src Port: 8080, Dst Port: 54524
▸ Data (25 bytes)
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 35 8b b4 40 00  40 11 96 e6 0a 00 02 0f   E··5··@· @·......
0020  0a 00 02 0f 1f 90 d4 fc  00 21 18 50 48 6f 6c 61   ........ ·!·PHola
0030  20 79 6f 20 73 6f 79 20  65 6c 20 73 65 72 76 69    yo soy  el servi
0040  64 6f 72 0a 00                                    dor··
```

Figura 9. Trama del servidor

Protocolo IP

Puerto fuente

Protocolo UDP

Puerto destino

IP fuente

Mensaje

IP destino

2.6 MENCIONA GRAFICAMENTE Y POR ESCRITO COMO CONFIGURASTE LOS FILTROS DEL WIRESHARK PARA RECIBIR LAS TRAMAS DEL CLIENTE Y DEL SERVIDOR.

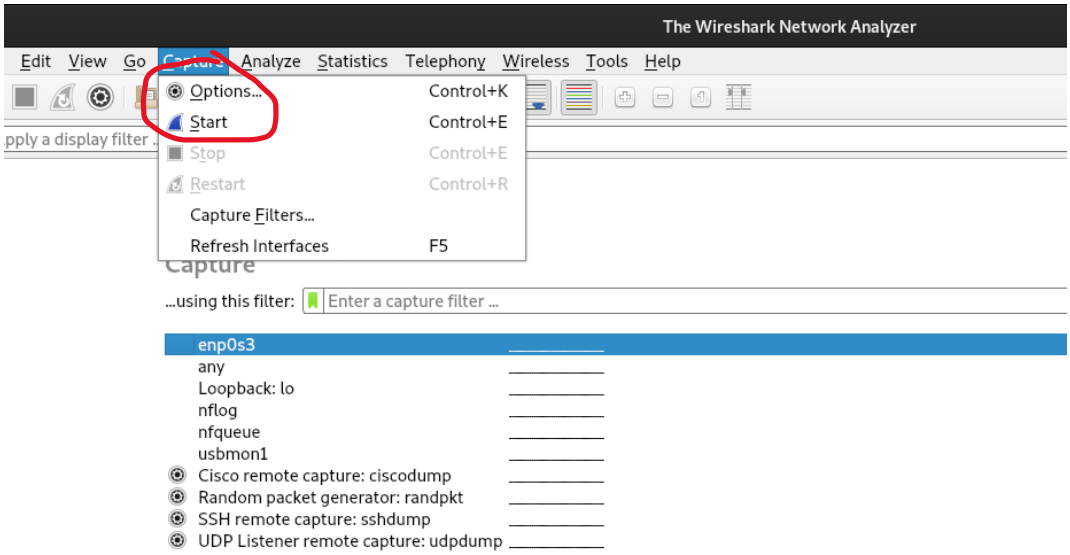

Figura 10. Pantalla inicial de Wireshark

Primeramente, al abrir wireshark nos dirigimos a options como se ve en la figura 10.

Una vez en la pestaña de options, habilitamos la opción de interfaz llamada any, ya que haremos la comunicación de manera local en nuestra máquina y damos en start, esto se ve en la figura 11.

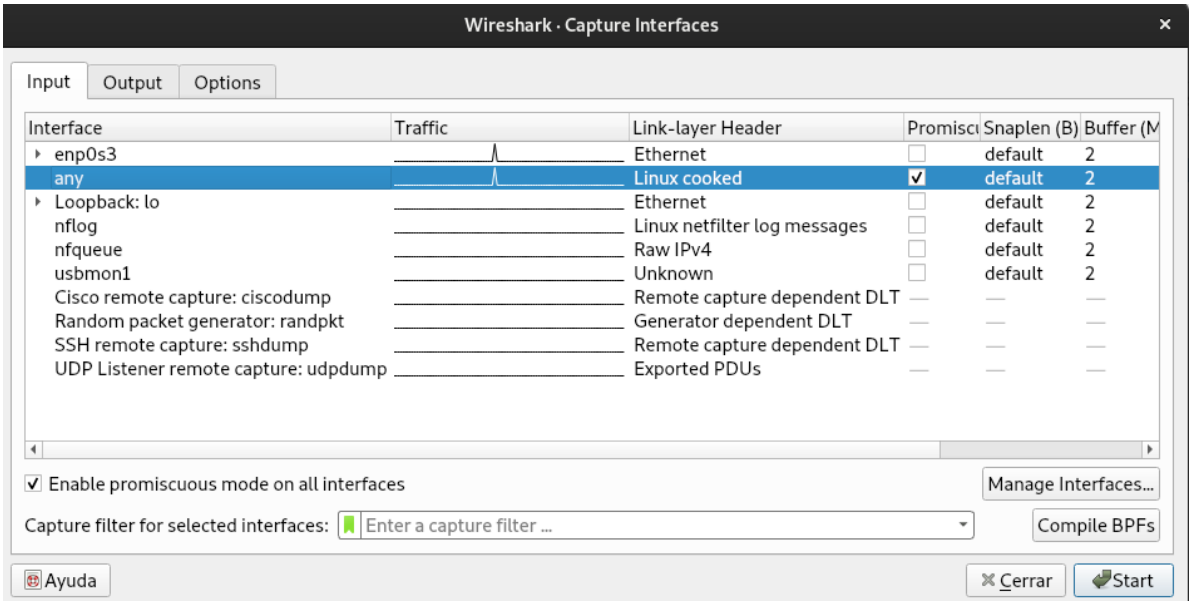

Figura 11. Opciones de wireshark

Finalizando con la configuración ponemos el filtro udp.port==8080, esto lo ponemos para capturar el trafico en ese puerto en específico ya que fue el especificado en los programas, esto lo vemos en la figura 12.

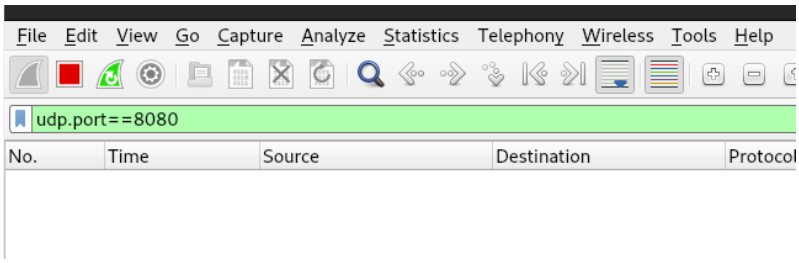

Figura 12. Filtro para el puerto

De esta forma ya podemos seguir el tráfico en ese puerto.

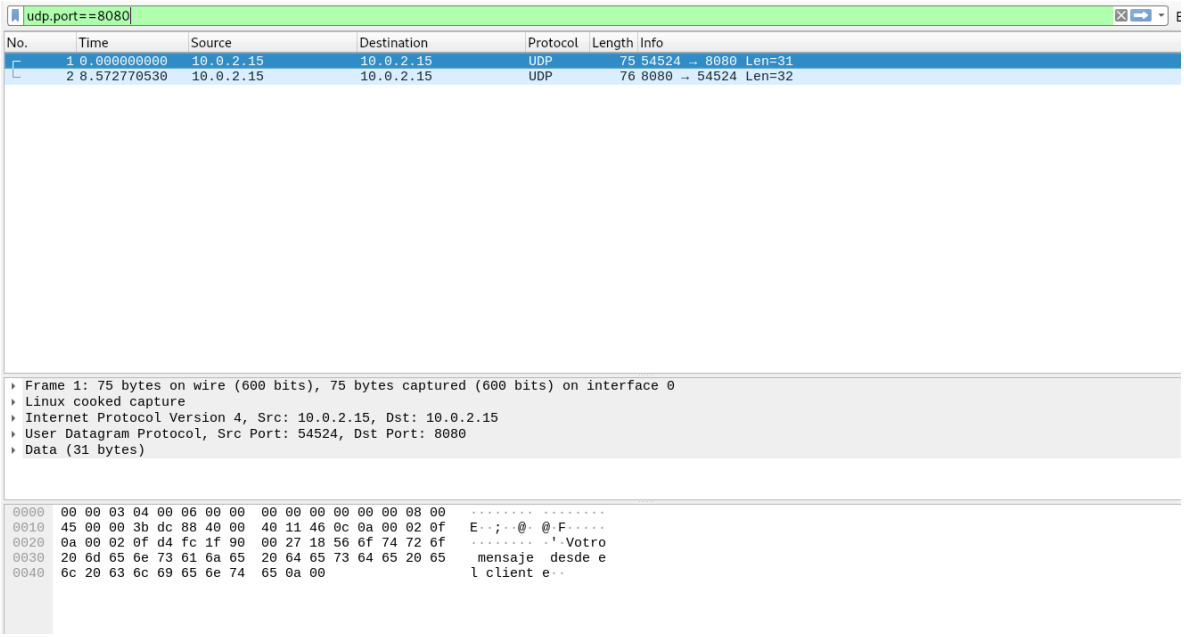

Figura 13. Tráfico capturado en wireshark

3. CONCLUSIONES INDIVIDUALES DE CADA PARTICIPANTE DEL EQUIPO

FISCHER SALAZAR CÉSAR EDUARDO

En esta práctica puede entender un poco más el cómo se realiza la comunicación entre un cliente y un servidor mediante la creación de un pequeño chat muy básico que realizamos a partir de la utilización de creación del programa servidor, así como una modificación en del programa del socket que teníamos para que este nos permitirá tener comunicación bidireccional entre ambos.

LÓPEZ GARCÍA JOSÉ EDUARDO

Por medio de la práctica 3, se ha comprendido un reforzamiento del tema de sockets y se pudo lograr una mejora del programa realizado en la práctica anterior, donde ahora se ha realizado la interacción del cliente-servidor a través de una especie de entorno de mensajería, donde se implementó un ciclo en el que pudieran hacer el envío de mensajes entre los participantes; con esto, se sentó una base importante para la realización del proyecto final que se tiene contemplado para esta materia, y fue interesante ver cómo se logró la forma en que se dio la comunicación entre estos de forma simulada.

MEZA VARGAS BRANDON DAVID

Con la presente práctica logramos implementar la base de nuestro proyecto final, haciendo una mejora al programa de la practica anterior, en este caso se implementó un ciclo while para que un cliente, así como un servidor, reciban y envíen mensajes uno a otro, creando así un pequeño chat.

Sin duda una muy buena práctica donde de igual forma, hicimos uso de wireshark, reforzando así la parte de identificar cada parte de la trama, personalmente, con esta práctica se me hizo más sencillo que la pasada, pues ya tenía conocimiento.