TP 1 - PARTE B

1) **Utilizando el código raw.c como base escribir un "sniffer" que es un programa que muestra el contenido del tráfico que llega.**

Código del cliente:

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8088
#define SA struct sockaddr
void func(int sockfd)
{
    int fact,hijo ,padre;
    double tiempo_t;
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Ingrese texto : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        if ((strncmp(buff, "SALIR", 5)) == 0) {
            printf("Saliendo...\n");
            break;
        }
        bzero(buff, sizeof(buff));
        read(sockfd, &fact, sizeof(fact));
        printf("Servidor : %d\n", fact);
        read(sockfd,(void*)&tiempo_t, sizeof(tiempo_t));
        printf("Tiempo de respuesta del servidor: %f microsegundos\n",
tiempo_t);
        read(sockfd,(void*)&padre,sizeof(padre));
```

```
        read(sockfd,(void*)&hijo,sizeof(hijo));
        printf("PID Padre:%d     PID Hijo:%d\n", padre,hijo);

        if ((strncmp(buff, "SALIR", 5)) == 0) {
            printf("Saliendo...\n");
            exit(0);
        }
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket: creo socket y lo verifico
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado ..\n");
    bzero(&servaddr, sizeof(servaddr));

    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // Conecto los sockets entre cliente y servidor
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
        != 0) {
        printf("Falla de conexión con servidor...\n");
        exit(0);
    }
    else
        printf("Conectado al servidor..\n");

    // Función para el chat
    func(sockfd);

    //Cierro el socket
```

```
    close(sockfd);
}
```

Código del servidor:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#include <arpa/inet.h>
#include <time.h>
#include <stdlib.h>
#define MAX 80
#define PORT 8088
#define SA struct sockaddr

/*PARA USAR: CUANDO ESTABLEZCO CADA CONEXION, MNADO MENSAJE A SERVER Y
SERVER LE CONTESTA, Y LUEGO PUEDO MANDAR MENSAJE DESDE OTRO CLIENTE.
PARA QUE FUNCIONE A CADA MENSAJE QUE MANDO AL SERVER, PRIMERO LE
CONTESTO Y LUEGO PUEDO MANDAR MENSAJE DESDE OTRO CLEINTE
PUEDO CERRAR LAS CONEXIONES DESDE EL CLIENTE, CUANDO ESTAN TODAS
CERRADAS SE CIERRA AUTOMATICAMENTE EL SERVIDOR
*/
/*struct timeval {
    int tv_sec; // segundos
    int tv_usec; // microsegundos
};*/
double tiempo_transcurrido(struct timespec* inicio, struct timespec*
fin) {
    return (fin->tv_sec - inicio->tv_sec) + (fin->tv_nsec -
inicio->tv_nsec) ;
}
// Función principal
int main(){
    //1 parte de agregado para multi
    fd_set master, read_fds;
    int fdmax,yes=1,active_connections=0;
    int i,j;
```

```
    FD_ZERO(&master);
    FD_ZERO(&read_fds);
    char buff[MAX];
    int n;
    struct timespec start,end;
    double tiempo_t;

    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado...\n");
    //bzero(&servaddr, sizeof(servaddr));

    //2 parte de agregado para multi
    if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))==
-1){
        perror("setsockopt");
        exit(0);
    }

    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    //3 parte agregado para multi
    memset(&(servaddr.sin_zero),'\0',8);

    // Bind del nuevo socket a la dirección IP y lo verifico(asocio ip
a puerto)
    if ((bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)))
== -1) {
        printf("Falla socket bind ...\n");
        exit(0);
    }
    else
```

```
        printf("Se hace el socket bind ..\n");

    // El servidor está en modo escucha (pasivo) y lo verifico
    if ((listen(sockfd, 5)) == -1) {
        printf("Falla el listen ...\n");
        exit(0);
    }else{
        printf("Servidor en modo escucha ...\n");
        //4 parte agregado para multi
        FD_SET(sockfd,&master);
        fdmax=sockfd;
    }
    len = sizeof(cli);

    //5 parte de agregado para multi
    // Rest of your code...

//5 parte de agregado para multi
     for (;;) {
        read_fds = master;
        if (select(fdmax + 1, &read_fds, NULL, NULL, NULL) == -1) {
            perror("select");
            continue;
        }

        for (i = 0; i <= fdmax; i++) {
            if (FD_ISSET(i, &read_fds)) {
                if (i == sockfd) {
                    // New connection request
                    len = sizeof(cli);
                    connfd = accept(sockfd, (struct sockaddr*)&cli,
(unsigned int*)&len);
                    if (connfd == -1) {
                        perror("accept");
                    } else {
                        // Add new connection to master set
                        FD_SET(connfd, &master);
                        if (connfd > fdmax) {
                            fdmax = connfd;
                        }
                        printf("Nuevo cliente conectado desde %s en el
socket %d\n", inet_ntoa(cli.sin_addr), connfd);
                        active_connections++;
```

```
                    }
                } else {
                    // Existing connection has data to read
                    bzero(buff, MAX);
                    clock_gettime(CLOCK_MONOTONIC,&start);
                    if ((n = recv(i, buff, sizeof(buff), 0)) <= 0) {
                        if (n == 0) {
                            printf("Cliente %d se desconectó\n", i);
                            active_connections--;
                        } else {
                            perror("recv");
                        }
                        // Close the connection and remove from master
set
                        close(i);
                        FD_CLR(i, &master);
                    } else {
                        // Print received message from client
                        printf("Del cliente %d: %s\n", i, buff);

                        // Check if the message is "SALIR" to close the
connection
                        if (strncmp("SALIR", buff, 5) == 0) {
                            printf("Cliente %d solicita salir\n", i);
                            active_connections--;
                            close(i);
                            FD_CLR(i, &master);
                            if(active_connections==0){
                                printf("Se cerraron todas las
conexiones.\nSalgo del servidor...\n");
                                exit(0);
                            }
                        } else {
                            int padre=getpid(),hijo=getppid();
                            int num = atoi(buff);
                            int factorial = 1;
                            for (int k = 1; k <= num; ++k) {
                                factorial *= k;
                            }
                            // Get input message from server and send
to client
                            printf("Servidor: %d\n",factorial);
                            bzero(buff, MAX);
```

```
                            n = 0;
                            write(i, (void*)&factorial,
sizeof(factorial));
                            clock_gettime(CLOCK_MONOTONIC,&end);
                            tiempo_t=tiempo_transcurrido(&start,&end);
                            write(i,&tiempo_t,sizeof(tiempo_t));
                            write(i,&padre,sizeof(padre));
                            write(i,&hijo,sizeof(hijo));
                        }
                    }
                }
            }
        }
    }

    // Close the socket when finished
    close(sockfd);
}
```

Utilizando como base el código del archivo “raw_socket_sniffer.c”, se filtran los paquetes para imprimir solo los correspondientes al puerto 8088 ya que es el puerto es el utilizado por el cliente y el servidor para la conexión.

Código del raw socket sniffer:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>

#define BUFFER_SIZE 65536

void procesar_paquete(unsigned char *buffer, int size) {
    struct iphdr *encabezado_ip = (struct iphdr *)buffer;
    unsigned short longitud_encabezado_ip = encabezado_ip->ihl * 4;

    struct sockaddr_in source, dest;
    memset(&source, 0, sizeof(source));
```

```
    memset(&dest, 0, sizeof(dest));
    source.sin_addr.s_addr = encabezado_ip->saddr;
    dest.sin_addr.s_addr = encabezado_ip->daddr;

    if (encabezado_ip->protocol == IPPROTO_TCP) {
        struct tcphdr *encabezado_tcp = (struct tcphdr *)(buffer +
longitud_encabezado_ip);
        unsigned int puerto_origen = ntohs(encabezado_tcp->source);
        unsigned int puerto_destino = ntohs(encabezado_tcp->dest);
        if(puerto_destino==8088|| puerto_origen==8088){
            printf("Paquete TCP - Dirección IP de origen: %s, Puerto de
origen: %u, Dirección IP de destino: %s, Puerto de destino: %u\n",
            inet_ntoa(source.sin_addr), puerto_origen,
inet_ntoa(dest.sin_addr), puerto_destino);
        }

    } else if (encabezado_ip->protocol == IPPROTO_UDP) {
        struct udphdr *encabezado_udp = (struct udphdr *)(buffer +
longitud_encabezado_ip);
        unsigned int puerto_origen = ntohs(encabezado_udp->source);
        unsigned int puerto_destino = ntohs(encabezado_udp->dest);
        if(puerto_destino==8088 || puerto_origen==8088){
            printf("Paquete UDP - Dirección IP de origen: %s, Puerto de
origen: %u, Dirección IP de destino: %s, Puerto de destino: %u\n",
            inet_ntoa(source.sin_addr), puerto_origen,
inet_ntoa(dest.sin_addr), puerto_destino);
        }

    } else if (encabezado_ip->protocol == IPPROTO_ICMP) {
        printf("Paquete ICMP - Dirección IP de origen: %s, Dirección IP
de destino: %s\n",
            inet_ntoa(source.sin_addr), inet_ntoa(dest.sin_addr));
    } else {
        printf("Paquete de protocolo desconocido\n");
    }
}

int main() {
    int sockfd;
    unsigned char buffer[BUFFER_SIZE];

    // Crear un socket raw
    if ((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_TCP)) < 0) {
```

```
        perror("socket");
        exit(EXIT_FAILURE);
    }

    // Recibir paquetes
    while (1) {
        int bytes_recibidos = recvfrom(sockfd, buffer, sizeof(buffer),
0, NULL, NULL);
        if (bytes_recibidos < 0) {
            perror("recvfrom");
            exit(EXIT_FAILURE);
        }

        procesar_paquete(buffer, bytes_recibidos);
    }

    return 0;
}
```

2) Enviar tráfico al "sniffer" desde el cliente escrito en la parte A del TP1

Output del sniffer:

Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 8088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 44616
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 8088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 44616
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 8088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 44616
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 8088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 44616
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088

Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 8088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 44616
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 8088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 44616
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 44616, Dirección IP de destino: 127.0.0.1, Puerto de destino: 8088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 8088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 44616

Output del servidor:

Socket creado...
Se hace el socket bind ..
Servidor en modo escucha ...
Nuevo cliente conectado desde 127.0.0.1 en el socket 4
Del cliente 4: hola

Servidor: 1
Del cliente 4: 5

Servidor: 120

Output del cliente:

Socket creado ..
Conectado al servidor..
Ingrese texto : 5
Servidor : 120
Tiempo de respuesta del servidor: 101000.000000 microsegundos
PID Padre:1584 PID Hijo:1561
Ingrese texto :

3) **Enviar tráfico ICMP al "sniffer" y mostrar los resultados del LOG con comentarios.**

Para enviar tráfico ICMP, se modificó el sniffer cambiando el protocolo de IPPROTO_TCP a IPPROTO_ICMP. Luego se utilizó el comando ping en la consola.

Output de la consola:

ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.150 ms

Output del sniffer:

Paquete ICMP - Dirección IP de origen: 127.0.0.1, Dirección IP de destino: 127.0.0.1
Paquete ICMP - Dirección IP de origen: 127.0.0.1, Dirección IP de destino: 127.0.0.1
Paquete ICMP - Dirección IP de origen: 127.0.0.1, Dirección IP de destino: 127.0.0.1