Protocolos de Internet

1° CUATRIMESTRE DE 2024

Trabajo Práctico Nº 1B

Grupo: 4

Profesor/a: Javier Adolfo Ouret

Integrantes:

N°	Apellido y Nombre	Legajo	Email
1	Marina Mercadal	152150976	marinamercadal@uca.edu,ar
2	Martina Naiquen Ruiz	29181	martinaruiz@uca.edu.ar
3	Carolina Suarez	152000738	suarezmacaro@gmail.com

Corrección:

Entrega 1	Devolución 1	Entrega 2	Nota

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

A base de el código brindado en clase, se crea un socket raw con tcp (`IPPROTO_TCP`)

```
// Crear 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);
    }
```

El código completo

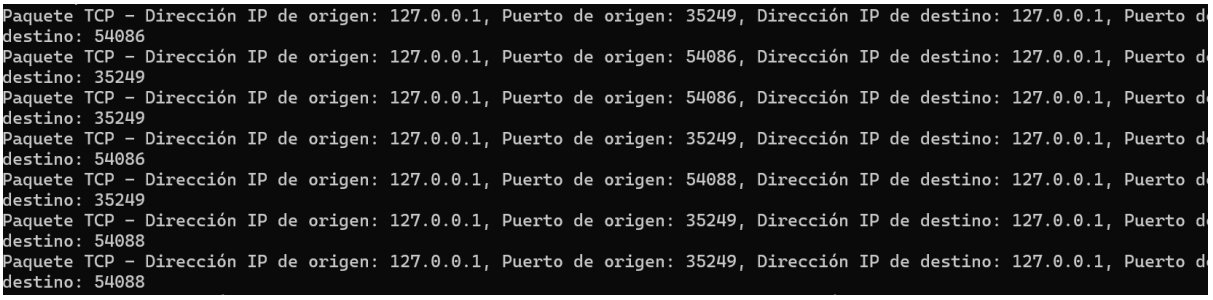
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>

#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;

    // Obtener las direcciones IP de origen y destino
    struct in_addr addr_origen, addr_destino;
    addr_origen.s_addr = encabezado_ip->saddr;
```

```
    addr_destino.s_addr = encabezado_ip->daddr;

    // Mostrar la información del paquete
    printf("Origen: %s, Destino: %s, Tamaño del paquete: %d bytes\n",
           inet_ntoa(addr_origen), inet_ntoa(addr_destino), size);
    printf("Contenido del paquete en formato hexadecimal:\n");

    // Mostrar el contenido del paquete en formato hexadecimal
    for (int i = 0; i < size; ++i) {
        printf("%02x ", buffer[i]);
        if ((i + 1) % 16 == 0) {
            printf("\n");
        }
    }
    printf("\n\n");
}

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

    // Crear 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;
}
```

Salida

```
Origen: 127.0.0.1, Destino: 127.0.0.1, Tamaño del paquete: 52 bytes
Contenido del paquete en formato hexadecimal:
45 00 00 34 2d b0 40 00 40 06 0f 12 7f 00 00 01
7f 00 00 01 a3 0b c0 08 f7 0d 66 41 95 1e c4 6d
80 10 01 ff fe 28 00 00 01 01 08 0a 9e 8d c9 a7
9e 8d c9 a7

Origen: 127.0.0.1, Destino: 127.0.0.1, Tamaño del paquete: 90 bytes
Contenido del paquete en formato hexadecimal:
45 00 00 5a 2d b1 40 00 40 06 0e eb 7f 00 00 01
7f 00 00 01 a3 0b c0 08 f7 0d 66 41 95 1e c4 6d
80 18 02 00 fe 4e 00 00 01 01 08 0a 9e 8d c9 a9
9e 8d c9 a7 82 24 01 00 00 66 fc 00 00 48 9b 00
00 00 05 05 00 00 2c 86 01 00 00 66 fd 00 00 48
9b 00 00 00 05 07 00 00 2c 86

Origen: 127.0.0.1, Destino: 127.0.0.1, Tamaño del paquete: 52 bytes
Contenido del paquete en formato hexadecimal:
45 00 00 34 30 74 40 00 40 06 0c 4e 7f 00 00 01
7f 00 00 01 c0 08 a3 0b 95 1e c4 6d f7 0d 66 67
80 10 02 00 fe 28 00 00 01 01 08 0a 9e 8d c9 a9
9e 8d c9 a9

Origen: 127.0.0.1, Destino: 127.0.0.1, Tamaño del paquete: 71 bytes
Contenido del paquete en formato hexadecimal:
45 00 00 47 89 a9 40 00 40 06 b3 05 7f 00 00 01
7f 00 00 01 c0 0a a3 0b f5 3c 53 4f 32 c7 49 c2
80 18 27 f9 fe 3b 00 00 01 01 08 0a 9e 8d cb 1a
9e 8d c9 a3 82 8d de 12 f0 b7 d7 12 f0 b7 de 12
f0 c7 8d 12 f0 b7 de

Origen: 127.0.0.1, Destino: 127.0.0.1, Tamaño del paquete: 52 bytes
Contenido del paquete en formato hexadecimal:
45 00 00 34 11 b6 40 00 40 06 2b 0c 7f 00 00 01
7f 00 00 01 a3 0b c0 0a 32 c7 49 c2 f5 3c 53 62
80 10 02 00 fe 28 00 00 01 01 08 0a 9e 8d cb 1a
9e 8d cb 1a
```

Aca podemos ver las direcciones ip de origen, destino y el tamaño del paquete, junto a su contenido.



Aca podemos ver además de las direcciones ip de origen, destino y el tamaño del paquete, el puerto de origen y destino.

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

Código del sniffer:

```c
#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);
```

```
        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);

        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_ICMP)) < 0) {
//IPPROTO_ICMP IPPROTO_TCP
        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;
}
```

A partir de lo hecho en el TP-A de cliente y servidor, se les asignó a ambos un puerto específico. En nuestro caso asignamos el puerto 9000.

```
#define MAX 80
#define PORT 9000
```

Se ejecutó primero al sniffer desde la terminal de ubuntu, se prosiguió ejecutando al servidor y luego al cliente. Una vez con el servidor y el cliente conectados y el sniffer corriendo, se buscó al puerto (9000) desde la terminal de ubuntu.

```
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 35249, Dirección IP de destino: 127.0.0.1, Puerto de destino: 54086
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 54086, Dirección IP de destino: 127.0.0.1, Puerto de destino: 35249
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 54086, Dirección IP de destino: 127.0.0.1, Puerto de destino: 35249
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 54088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 35249
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 9000, Dirección IP de destino: 127.0.0.1, Puerto de destino: 37070
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 37070, Dirección IP de destino: 127.0.0.1, Puerto de destino: 9000
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 35249, Dirección IP de destino: 127.0.0.1, Puerto de destino: 54088
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 54088, Dirección IP de destino: 127.0.0.1, Puerto de destino: 35249
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 35249, Dirección IP de destino: 127.0.0.1, Puerto de destino: 54086
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 54086, Dirección IP de destino: 127.0.0.1, Puerto de destino: 35249
Paquete TCP - Dirección IP de origen: 127.0.0.1, Puerto de origen: 54086, Dirección IP de destino: 127.0.0.1, Puerto de destino: 35249
```

Acá podemos ver, en la línea de terminal subrayada, que el puerto de origen coincide con el de nuestro cliente-servidor. En la línea de abajo , el puerto de destino también tiene al puerto 9000.

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

Para lograr este punto, realizamos modificaciones en nuestro sniffer.c

```
// Crear un socket raw
if ((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) < 0) {
    perror("socket");
    exit(EXIT_FAILURE);
}
```

Para esto, cambiamos el tercer parámetro del socket, en donde en vez de poner `IPPROTO_TCP,` se puso `IPPROTO_ICMP`

Luego proseguimos realizando un “ping” a 2 direcciones de ip:

ping 127.0.0.1

```
marina@LAPTOP-2O4JRJF2:~/.vscode-server/.vscode/TP1B$ 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=0.036 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.045 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.036 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.035 ms
^C
--- 127.0.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9371ms
rtt min/avg/max/mdev = 0.034/0.038/0.046/0.004 ms
```

ping 127.0.0.2

```
marina@LAPTOP-2O4JRJF2:~/.vscode-server/.vscode/TP1B$ ping 127.0.0.2
PING 127.0.0.2 (127.0.0.2) 56(84) bytes of data.
64 bytes from 127.0.0.2: icmp_seq=1 ttl=64 time=0.049 ms
64 bytes from 127.0.0.2: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 127.0.0.2: icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from 127.0.0.2: icmp_seq=4 ttl=64 time=0.040 ms
64 bytes from 127.0.0.2: icmp_seq=5 ttl=64 time=0.037 ms
64 bytes from 127.0.0.2: icmp_seq=6 ttl=64 time=0.030 ms
64 bytes from 127.0.0.2: icmp_seq=7 ttl=64 time=0.072 ms
64 bytes from 127.0.0.2: icmp_seq=8 ttl=64 time=0.047 ms
64 bytes from 127.0.0.2: icmp_seq=9 ttl=64 time=0.035 ms
64 bytes from 127.0.0.2: icmp_seq=10 ttl=64 time=0.035 ms
^C
--- 127.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9346ms
rtt min/avg/max/mdev = 0.030/0.042/0.072/0.011 ms
```