# Desarrollo

## Cliente

El cliente realizará una conexión hacia el servidor utilizando un socket. El proceso es muy similar a lo que realizamos en la práctica 1, incluso cuando el lenguaje de programación sea distinto, muchos de los métodos tienen nombres similares.

El cliente crea un socket e intenta realizar una conexión hacia localhost (o 127.0.0.1) en el puerto 5555 que nosotros definimos.

Posteriormente procede a mandar un mensaje hacia el servidor, en este caso la cadena de caracteres “Hola desde el cliente”. Espera la respuesta del servidor y la imprime, para finalmente cerrar el socket y terminar el programa.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <fcntl.h>

#include <sys/socket.h>

#define PORT 5555

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char \*hello = "Hola desde el cliente";

char buffer[BUFFER\_SIZE] = {0};

// Crear socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation error");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convertir direcciones IPv4 e IPv6 de texto a binario

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

perror("Invalid address/ Address not supported");

return -1;

}

// Conectarse al servidor

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Connection Failed");

return -1;

}

// Enviar mensaje al servidor

send(sock, hello, strlen(hello), 0);

printf("Mensaje enviado\n");

// Leer respuesta del servidor

read(sock, buffer, BUFFER\_SIZE);

printf("Respuesta del servidor: %s\n", buffer);

// Cerrar el socket

close(sock);

return 0;

}

Bloque de código 1. Código del cliente.

## Servidor

El servidor en este caso se encarga de crear un socket para aceptar conexiones, en este caso el socket se configura en modo no bloqueante pasando la constante ***O\_NONBLOCK***, también se configura el socket para recibir conexiones de entrada desde cualquier IP pero que la conexión llegue en específico al puerto 5555 de la computadora donde se ejecuta el servidor.

Procede a escuchar conexiones a través de ese socket (a lo mucho recibe hasta 30 sockets entrantes para conexión) y por medio de un ciclo ***while*** infinito acepta y maneja esas conexiones.

En ese ciclo el servidor espera actividad, si detecta alguna significa que es una nueva conexión entrante, por lo que crea un socket de conexión aceptando dicha conexión, e imprime los datos del origen de dicha conexión con su FD (file descriptor), su IP y el puerto de origen y agrega el socket entrante al arreglo de sockets.

Posteriormente maneja las entradas y salidas de los otros sockets: en un proceso donde primero revisa si el socket fue cerrado y lee el mensaje que mandó el socket, si no ha sido cerrado el socket de conexión entrante, reenvía el mensaje que el cliente mandó al propio cliente.

El programa termina cuando el usuario force la terminación de la ejecución del programa, o cuando ocurra alguna excepción que no esté manejada.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <fcntl.h>

#include <sys/select.h>

#define PORT 5555

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket, max\_sd, sd, activity, valread;

int client\_socket[30] = {0};

int max\_clients = 30;

int addrlen;

struct sockaddr\_in address;

char buffer[BUFFER\_SIZE];

fd\_set readfds;

// Crear socket del servidor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Configurar el socket del servidor como no bloqueante

fcntl(server\_fd, F\_SETFL, O\_NONBLOCK);

// Configurar el tipo de socket

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Adjuntar el socket al puerto 5555

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Escuchar en el socket

if (listen(server\_fd, 3) < 0) {

perror("listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Escuchando en el puerto %d \n", PORT);

// Bucle principal para aceptar y manejar conexiones

while (1) {

// Limpiar el conjunto de descriptores de socket

FD\_ZERO(&readfds);

// Añadir el socket del servidor al conjunto de descriptores

FD\_SET(server\_fd, &readfds);

max\_sd = server\_fd;

// Añadir los sockets de cliente al conjunto de descriptores

for (int i = 0; i < max\_clients; i++) {

sd = client\_socket[i];

if (sd > 0)

FD\_SET(sd, &readfds);

if (sd > max\_sd)

max\_sd = sd;

}

// Esperar a que ocurra alguna actividad en uno de los sockets

activity = select(max\_sd + 1, &readfds, NULL, NULL, NULL);

if (activity < 0) {

perror("select error");

}

// Si hay una actividad en el socket del servidor, es una nueva conexión

if (FD\_ISSET(server\_fd, &readfds)) {

addrlen = sizeof(address);

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen)) < 0) {

perror("accept failed");

exit(EXIT\_FAILURE);

}

printf("Nueva conexión, socket fd es %d, ip es : %s, puerto : %d\n", new\_socket, inet\_ntoa(address.sin\_addr), ntohs(address.sin\_port));

// Añadir el nuevo socket al array de sockets de cliente

for (int i = 0; i < max\_clients; i++) {

if (client\_socket[i] == 0) {

client\_socket[i] = new\_socket;

printf("Añadiendo a la lista de sockets como %d\n", i);

break;

}

}

}

// Manejar IO en otros sockets

for (int i = 0; i < max\_clients; i++) {

sd = client\_socket[i];

if (FD\_ISSET(sd, &readfds)) {

// Revisar si fue por cierre y leer el mensaje

if ((valread = read(sd, buffer, BUFFER\_SIZE)) == 0) {

// Alguien se desconectó, obtener detalles e imprimir

getpeername(sd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen);

printf("Host desconectado, ip %s, puerto %d\n", inet\_ntoa(address.sin\_addr), ntohs(address.sin\_port));

// Cerrar el socket y marcarlo como 0 en la lista

close(sd);

client\_socket[i] = 0;

} else {

// Poner terminador de cadena en el buffer y enviar mensaje de vuelta al cliente

buffer[valread] = '\0';

send(sd, buffer, strlen(buffer), 0);

}

}

}

}

return 0;

}

Bloque de Código 2. Código del servidor.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 1. Ejecución del servidor y del cliente en dos terminales a la par.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 2. Captura en Wireshark del paquete enviado del cliente al servidor con el mensaje “Hola desde el cliente”.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 3. Captura en Wireshark del paquete enviado desde el servidor al cliente, con el mismo mensaje que mandó el cliente.

A screenshot of a computer

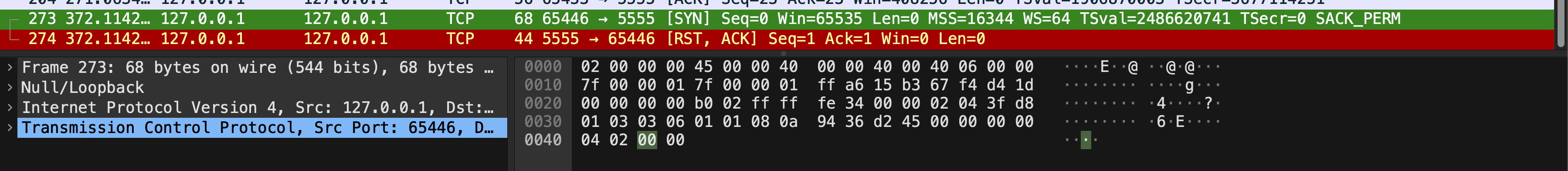
AI-generated content may be incorrect.  
Figura 4. Dos ejecuciones adicionales donde se puede ver que el puerto de origen de la conexión ha cambiado.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 5. Captura en Wireshark de los paquetes de esas dos ejecuciones adicionales.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 6. Ejecución del cliente cuando el servidor no está ejecutando.

  
Figura 7. Captura en Wireshark del intento de envió de paquete al servidor inactivo.