# Cliente

El cliente realiza la comunicación con el servidor a través del puerto 5555. Primero, crea un socket UDP y configura la dirección del servidor, especificando el protocolo IPv4, el puerto y permitiendo conexiones desde cualquier dirección IP. Si falla la creación del socket, el programa termina con un mensaje de error.

El cliente opera en un bucle infinito donde solicita al usuario un mensaje, lo envía al servidor mediante sendto() y espera una respuesta con recvfrom(). Antes de enviar el mensaje, elimina el salto de línea de la entrada del usuario para evitar problemas en la transmisión. Al recibir la respuesta del servidor, la imprime en pantalla y repite el proceso.

Dado que usa UDP, no hay establecimiento de conexión, y cada mensaje se maneja de forma independiente. El programa no tiene una condición de salida integrada, por lo que debe ser interrumpido manualmente (Ctrl+C). Este diseño es típico en aplicaciones cliente-servidor simples donde se prioriza la simplicidad sobre el manejo de errores avanzado o la terminación controlada.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 5555

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in servaddr;

ssize\_t n;

socklen\_t len;

// Crear el socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Error al crear el socket");

exit(EXIT\_FAILURE);

}

// Inicializar la dirección del servidor

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(PORT);

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

len = sizeof(servaddr);

while (1) {

printf("Ingrese el mensaje: ");

fgets(buffer, BUFFER\_SIZE, stdin);

// Eliminar el salto de línea

buffer[strcspn(buffer, "\n")] = '\0';

// Enviar el mensaje al servidor (bloqueante)

sendto(sockfd, buffer, strlen(buffer), 0, (const struct sockaddr \*)&servaddr, len);

// Recibir la respuesta del servidor (bloqueante)

n = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&servaddr, &len);

buffer[n] = '\0';

printf("Servidor : %s\n", buffer);

}

close(sockfd);

return 0;

}

# Servidor

Primero, crea un socket UDP y configura la dirección del servidor para aceptar conexiones en cualquier interfaz de red (INADDR\_ANY). Luego, enlaza el socket al puerto especificado usando bind(). Si alguna de estas operaciones falla, el programa termina con un mensaje de error.

El servidor funciona en un bucle infinito donde espera mensajes de clientes mediante recvfrom(), que es una llamada bloqueante. Cuando recibe un mensaje, lo imprime en la consola y envía la misma cadena de vuelta al cliente como respuesta usando sendto(). El servidor no tiene una condición de salida programada, por lo que debe ser detenido manualmente (Ctrl+C).

A diferencia de TCP, UDP no establece conexiones persistentes, por lo que cada mensaje se maneja de forma independiente. Este diseño es útil para aplicaciones simples donde no se requiere control de flujo o garantía de entrega, como servidores de eco (echo servers) o servicios de consulta-respuesta rápidos.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 5555

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in servaddr, cliaddr;

socklen\_t len;

ssize\_t n;

// Crear el socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Error al crear el socket");

exit(EXIT\_FAILURE);

}

// Inicializar la dirección del servidor

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

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

servaddr.sin\_port = htons(PORT);

// Enlazar el socket con la dirección del servidor

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) {

perror("Error en el bind");

close(sockfd);

exit(EXIT\_FAILURE);

}

len = sizeof(cliaddr);

while (1) {

// Recibir datos del cliente (bloqueante)

n = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&cliaddr, &len);

// Añadir un terminador de cadena al final del mensaje recibido

buffer[n] = '\0';

printf("Cliente : %s\n", buffer);

// Enviar una respuesta al cliente (bloqueante)

sendto(sockfd, buffer, n, 0, (struct sockaddr \*)&cliaddr, len);

}

close(sockfd);

return 0;

}

a