Sockets no orientados a conexiones bloqueantes

Los sockets no orientados a conexiones usan el protocolo UDP (User Datagram Protocol). UDP es un protocolo de comunicación de red que permite el envío de datagramas (paquetes de datos) sin necesidad de establecer una conexión previa entre el cliente y el servidor. Este tipo de socket no requiere una conexión continua o un proceso de "handshake" como en el protocolo TCP (Transmission Control Protocol).

Las características son:

- **Sin Conexión:** No hay una conexión establecida antes de enviar datos. Cada mensaje (datagrama) se envía y se recibe de forma independiente.
- **No Fiable:** UDP no garantiza la entrega de paquetes ni su orden, por lo que los paquetes pueden llegar desordenados, duplicados o incluso perderse.
- **Más Rápido:** Menos sobrecarga en comparación con TCP, ya que no hay necesidad de gestionar conexiones y asegurar la entrega de paquetes.
- **Bloqueante:** Por defecto, las operaciones de lectura y escritura en los sockets UDP son bloqueantes, es decir, el programa espera hasta que la operación se complete.

Para realizar esta práctica se debe seguir un flujo de un cliente y un servidor, siendo de forma inherente que el protocolo UDP es no orientado a conexión, no obstante, todo debe tener un flujo, para esta práctica se utiliza un cliente y servidor:

Cliente	Servidor
1. Crear Socket: Crear un socket UDP. 2. Enviar: Enviar un mensaje al servidor. 3. Recibir: Esperar y recibir la respuesta del servidor. 4. Repetir: Continuar enviando mensajes y recibiendo respuestas.	1. Crear Socket: Crear un socket UDP. 2. Enlazar: Asignar el socket a una dirección IP y un puerto. 3. Recibir: Escuchar mensajes de clientes. 4. Responder: Enviar respuestas a los clientes. 5. Repetir: Continuar recibiendo y respondiendo mensajes.

El flujo de esta práctica es pequeño, solo demuestra el funcionamiento de como mandar mensajes de un cliente a un servidor utilizando el esquema no orientado a conexiones bloqueantes.

Para esta práctica se crean dos archivos:

- Cliente.c
- Servidor.c

El cliente realiza las siguientes acciones utilizando sockets UDP. Primero, crea un socket UDP con la función socket(), especificando el dominio de comunicación AF_INET (IPv4) y el tipo SOCK_DGRAM (UDP). Luego, configura la dirección del servidor en la estructura sockaddr_in, estableciendo el puerto y la dirección IP (INADDR_ANY para escuchar en todas las interfaces disponibles). En un bucle infinito, el cliente lee un mensaje del usuario mediante fgets(), eliminando el salto de línea al final del mensaje. Envía este mensaje al servidor usando sendto(), especificando la dirección del servidor. Después, espera recibir una respuesta del servidor con recvfrom(), y muestra la respuesta en la consola. Las funciones

bloqueantes sendto() y recvfrom() garantizan que el cliente espere hasta que las operaciones de envío y recepción se completen. El código usa las bibliotecas <stdio.h>, <stdlib.h>, <string.h>, <unistd.h>, y <arpa/inet.h>.

Para la realización del cliente se agrega el siguiente fragmento de código en el archivo Cliente.c como se muestra a continuación:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#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);
        buffer[strcspn(buffer, "\n")] = '\0'; // Eliminar el salto de línea

        // 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;
}
```

Para el funcionamiento del servidor, al iniciarse, crea un socket y lo enlaza a un puerto específico para recibir mensajes. Utiliza recvfrom() para recibir datos de clientes y sendto() para responderles, operando en un bucle infinito que permite la comunicación continua. El socket está configurado para escuchar en cualquier dirección IP y el puerto 8080. Cada mensaje recibido se imprime y se envía de vuelta al cliente. El servidor no establece una conexión con los clientes, sino que simplemente envía y recibe datagramas de forma independiente, manteniéndose disponible para recibir nuevos mensajes en cualquier momento.

Para el servidor se debe de agregar el siguiente fragmento de código en el archivo servidor.c como se muestra a continuación:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#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);
        buffer[n] = '\0'; // Añadir un terminador de cadena al final del mensaje
recibido
        printf("Cliente : %s\n", buffer);

        // Enviar una respuesta al cliente (bloqueante)
        sendto(sockfd, buffer, n, 0, (struct sockaddr *)&cliaddr, len);
    }

    close(sockfd);
    return 0;
}
```

Para la ejecución de esta práctica primero debe de compilarse el par de archivos creados como se muestra a continuación:

gcc cliente.c -o cliente y gcc servidor.c -o servidor

La ejecución del programa se ve como se muestra a continuación: