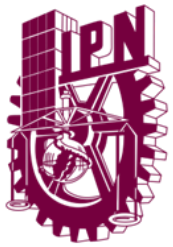


Instituto Politécnico Nacional

IPN

Escuela Superior de Cómputo

ESCOM

Academia de Redes

Aplicaciones para comunicaciones en red

6CV2

Práctica 4

“Sockets No Bloqueantes con UDP en C”

Alumna:

Navarrete Becerril Sharon Anette

Fecha de entrega: “Lunes, 21-Octubre-2024”

Profesor: Ojeda Santillan Rodrigo

OBJETIVO

El objetivo de esta práctica es implementar y demostrar el uso de sockets no bloqueantes orientados a datagramas utilizando el protocolo UDP (User Datagram Protocol) en un entorno cliente-servidor con el lenguaje de programación Java. Se busca que el estudiante sea capaz de enviar y recibir archivos de manera eficiente, utilizando operaciones de entrada y salida que no bloquean la ejecución del programa, optimizando así el manejo de múltiples conexiones en aplicaciones que requieren alta concurrencia, como la transmisión de archivos y servicios de mensajería.

INTRODUCCIÓN

Los sockets no bloqueantes orientados a datagramas (UDP) son una herramienta clave para aplicaciones que requieren la transmisión rápida y eficiente de datos sin necesidad de establecer una conexión previa, lo que es común en aplicaciones como juegos en línea y transmisión en tiempo real. A diferencia del protocolo TCP, que garantiza una comunicación fiable y ordenada, UDP no establece conexiones, permitiendo el envío de paquetes sin la verificación de su recepción o secuencia, lo que lo hace más rápido pero menos fiable.
En esta práctica, se busca implementar un cliente que envíe un archivo a un servidor utilizando sockets UDP no bloqueantes. El cliente será capaz de enviar los datos de manera continua sin bloquear el programa, mientras que el servidor recibirá y almacenará el archivo de manera no bloqueante, permitiendo la gestión eficiente de las operaciones de entrada y salida.

DESARROLLO

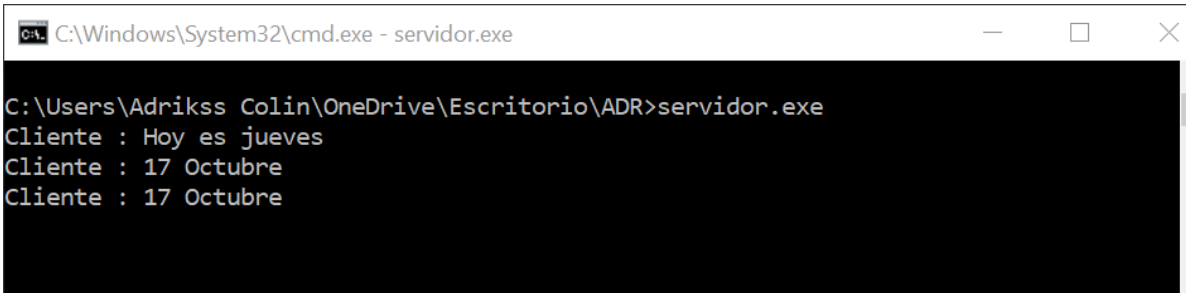
Se implementaron dos programas en Java, uno para el cliente y otro para el servidor. Ambos utilizan DatagramChannel para el manejo de los sockets UDP no bloqueantes, y el flujo general de la comunicación sigue el siguiente esquema:
Servidor
El servidor utiliza un canal UDP (DatagramChannel) no bloqueante para recibir los datagramas enviados por el cliente. Este canal está configurado con un Selector que permite manejar múltiples operaciones de entrada/salida de forma eficiente.
Los datos recibidos se almacenan en un ByteBuffer y luego se escriben en un archivo local denominado "archivorecibido.txt" usando un WritableByteChannel.
El servidor permanece en un bucle infinito, esperando recibir datagramas de los clientes y procesándolos en tiempo real.

Código del servidor:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>    // Necesario para memset y strlen
#include <winsock2.h>  // Necesario para sockets en Windows
#include <ws2tcpip.h>  // Necesario para funciones de red

#define PORT 8080
#define BUFFER_SIZE 1024
```

```
int main() {
    WSADATA wsaData;
    int sockfd;
    char buffer[BUFFER_SIZE];
    struct sockaddr_in servaddr, cliaddr;
    int len;
    int n;

    // Inicializar Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        perror("Error al inicializar Winsock");
        return EXIT_FAILURE;
    }

    // Crear el socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Error al crear el socket");
        WSACleanup();
        return EXIT_FAILURE;
    }

    // Inicializar la dirección del servidor
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;            // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;    // Aceptar conexiones de cualquier IP
    servaddr.sin_port = htons(PORT);          // Convertir el puerto al formato de red

    // Enlazar el socket con la dirección del servidor
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("Error en el bind");
        closesocket(sockfd);
        WSACleanup();
        return EXIT_FAILURE;
    }

    len = sizeof(cliaddr);

    while (1) {
        // Recibir datos del cliente (bloqueante)
        n = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr *)&cliaddr, &len);
        if (n < 0) {
            perror("Error al recibir datos");
            break;
        }

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

        // Enviar una respuesta al cliente (bloqueante)
        if (sendto(sockfd, buffer, n, 0, (struct sockaddr *)&cliaddr, len) < 0) {
            perror("Error al enviar respuesta");
            break;
        }
```

```
    }

    closesocket(sockfd);
    WSACleanup(); // Limpiar Winsock
    return 0;
}
```



Cliente

El cliente abre un canal UDP (`DatagramChannel`) en modo no bloqueante y configura la dirección del servidor. Utiliza un `ByteBuffer` para almacenar los datos del archivo que se desea enviar.

Los datos se leen del archivo en bloques y se envían al servidor mediante el método `send()`.

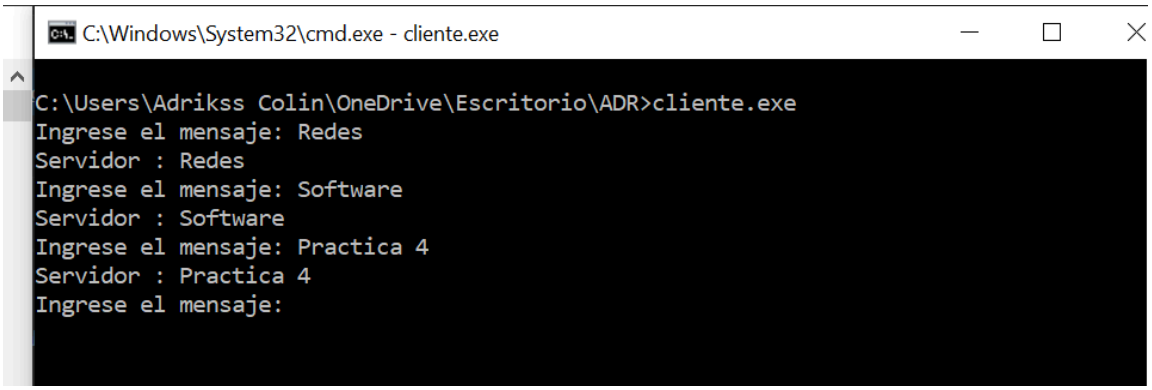
Una vez que se completa el envío del archivo, el canal se cierra.

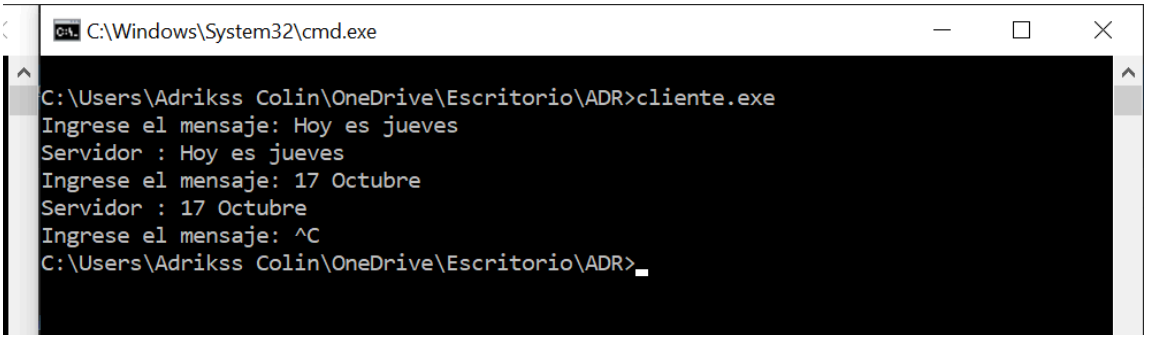
Código del cliente:

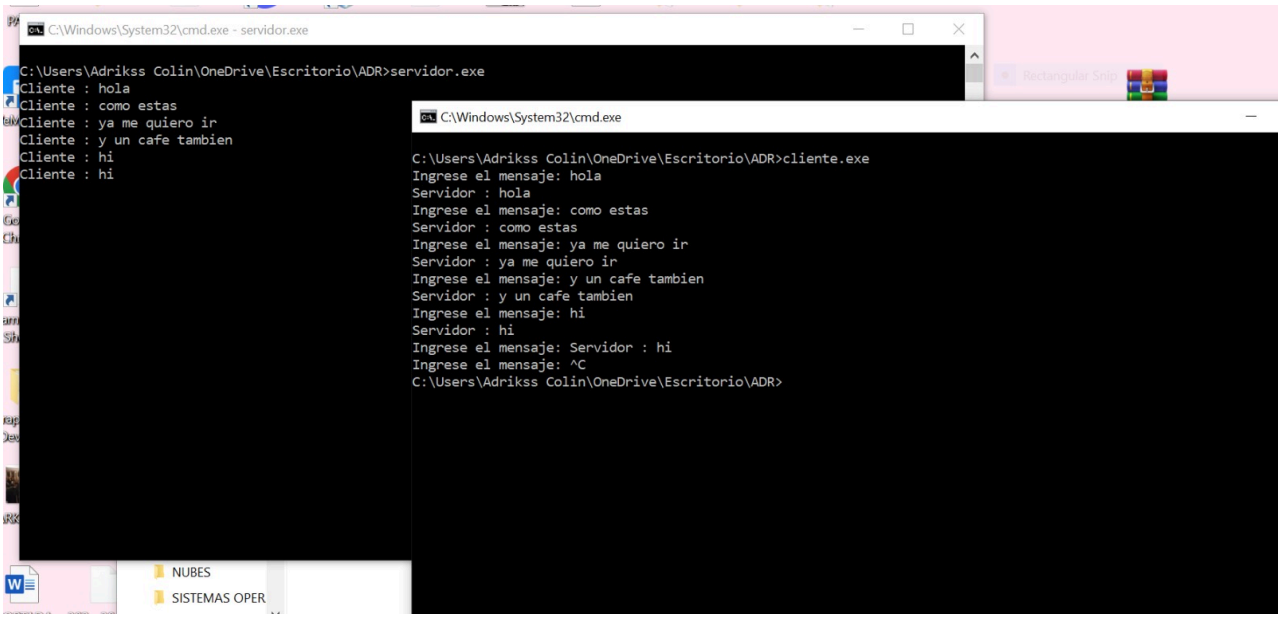
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#include <ws2tcpip.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
```

```
WSADATA wsaData;
int sockfd;
char buffer[BUFFER_SIZE];
struct sockaddr_in servaddr;
int n;
int len = sizeof(servaddr);

// Inicializar Winsock
if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
    perror("Error al inicializar Winsock");
    return EXIT_FAILURE;
}

// Crear el socket
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Error al crear el socket");
    WSACleanup();
    return EXIT_FAILURE;
}

// Inicializar la dirección del servidor
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(PORT);

// Asignar la dirección IP del servidor
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
if (servaddr.sin_addr.s_addr == INADDR_NONE) {
    perror("Error en inet_addr");
    closesocket(sockfd);
    WSACleanup();
    return EXIT_FAILURE;
}

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)
    if (sendto(sockfd, buffer, strlen(buffer), 0, (const struct sockaddr *)&servaddr, len) < 0) {
        perror("Error al enviar mensaje");
        closesocket(sockfd);
        WSACleanup();
        return EXIT_FAILURE;
    }

    // Recibir la respuesta del servidor (bloqueante)
    n = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr *)&servaddr, &len);
    if (n < 0) {
        perror("Error al recibir respuesta");
        closesocket(sockfd);
        WSACleanup();
        return EXIT_FAILURE;
```

```
        }
        buffer[n] = '\0';
        printf("Servidor : %s\n", buffer);
    }

    closesocket(sockfd);
    WSACleanup(); // Limpiar Winsock
    return 0;
}
```







CONCLUSIONES

Navarrete Becerril Sharon Anette:

Uso de sockets no bloqueantes optimiza la transmisión de archivos: La implementación de sockets UDP no bloqueantes permite enviar y recibir archivos de manera continua sin interrumpir la ejecución del programa. Esto es especialmente útil en aplicaciones que requieren alta concurrencia y transmisión en tiempo real, como servicios de mensajería y juegos en línea.

Bibliografía

Stevens, W. Richard. Programación en red con Unix: La API de Sockets. Addison-Wesley Professional, 2003.

Forouzan, Behrouz A. Comunicaciones de Datos y Redes. McGraw-Hill, 2012. Tanenbaum, Andrew S., y David J. Wetherall. Redes de Computadoras. Pearson, 2010.

Comer, Douglas E. Internetworking con TCP/IP Volumen Uno. Prentice Hall, 2006. Kurose, James F., y Keith W. Ross. Redes de Computadoras: Un Enfoque Descendente. Pearson, 2017.

Beazley, David, y Brian K. Jones. Python Cookbook: Recetas para Dominar Python 3. O'Reilly Media, 2013. Begg, A. Sockets TCP/IP en C: Guía Práctica para Programadores. Morgan Kaufmann, 2000