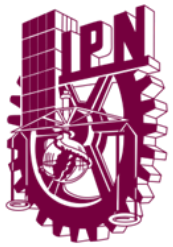


Instituto Politécnico Nacional

IPN

Escuela Superior de Cómputo

ESCOM

Academia de Redes

Aplicaciones para comunicaciones en red

6CV2

Práctica 1

“Sockets Orientados a Conexiones Bloqueantes”

Alumna

Navarrete Becerril Sharon Anette

Fecha de entrega: “Lunes, 14-Octubre-2024”

Profesor: Ojeda Santillan Rodrigo

OBJETIVO

El objetivo de esta práctica es comprender e implementar sockets orientados a conexiones bloqueantes dentro de un entorno cliente-servidor, utilizando el lenguaje de programación Python. Se busca establecer una conexión confiable y secuencial mediante el protocolo TCP (Transmission Control Protocol), donde las operaciones de red, como conectar, enviar y recibir datos, bloquean la ejecución del programa hasta completarse. Al finalizar la práctica, se espera que los estudiantes puedan establecer conexiones bloqueantes entre un cliente y un servidor, transmitir datos de forma efectiva, y analizar el tráfico de red generado utilizando Wireshark.

INTRODUCCIÓN

Los sockets orientados a conexiones bloqueantes representan una forma de comunicación en red que sigue el modelo cliente-servidor, basándose en el protocolo TCP, el cual garantiza una comunicación fiable y ordenada. En este tipo de sockets, las operaciones de red como conectar y enviar o recibir datos son bloqueantes, es decir, la ejecución del programa se detiene hasta que la operación se completa exitosamente. Esto los hace útiles en situaciones donde es crucial mantener el orden y la integridad de los datos transmitidos, como en aplicaciones de mensajería, transferencia de archivos o servicios web.

El uso de sockets bloqueantes garantiza que los datos lleguen al destino en el orden correcto, sin pérdida, lo que es especialmente importante en entornos donde se requieren conexiones fiables. Durante la práctica, se implementaron dos scripts en Python: uno para el cliente y otro para el servidor, los cuales se comunican utilizando sockets TCP en modo bloqueante. Además, se utilizó Wireshark para analizar el tráfico de red y observar en detalle el intercambio de paquetes.

DESARROLLO

Se desarrollaron dos scripts en Python, utilizando la librería socket, uno para el cliente y otro para el servidor. A continuación se describe la lógica de ambos programas:

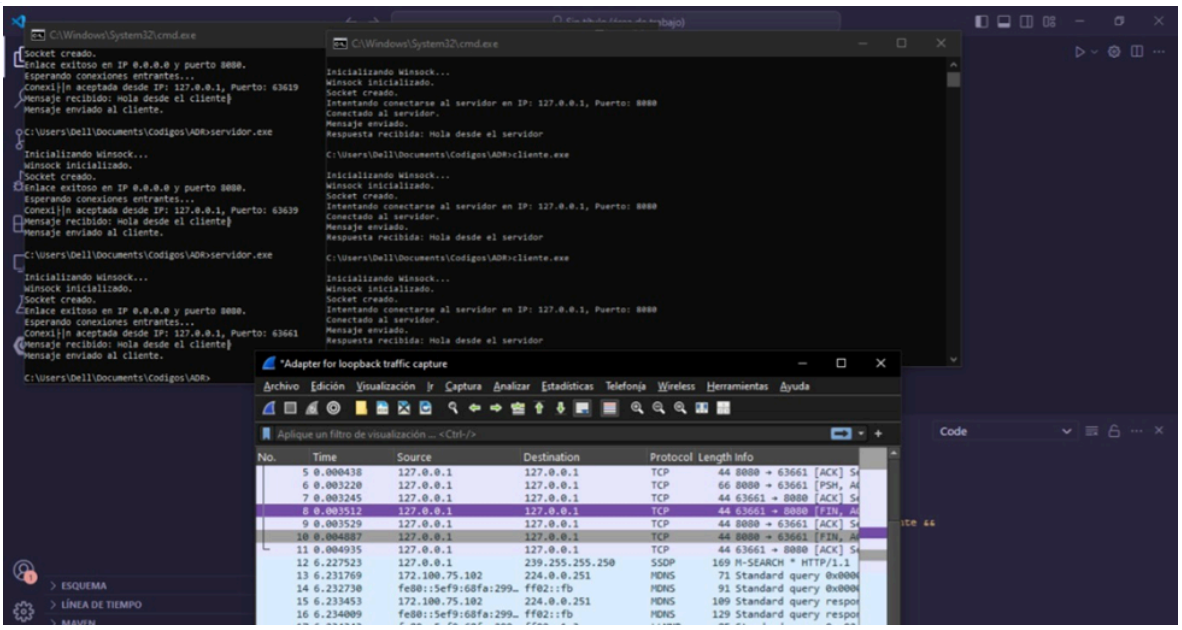
Servidor

El servidor se configura para escuchar conexiones en el puerto 8080. Utilizando el protocolo TCP, el servidor espera conexiones entrantes y, una vez que un cliente se conecta, acepta la conexión.

El servidor está diseñado en modo bloqueante, lo que significa que su ejecución se detiene hasta que un cliente intente conectarse. Una vez establecida la conexión, el servidor recibe el mensaje del cliente, lo muestra y luego cierra la conexión de manera controlada.

Código del servidor:

```
import socket

def servidor(host='127.0.0.1', puerto=8080):
    # Crear un socket TCP/IP
    servidor_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Establecer el socket como bloqueante
    servidor_socket.setblocking(True)

    # Vincular el socket al puerto
    servidor_socket.bind((host, puerto))

    # Escuchar conexiones entrantes
    servidor_socket.listen(1)
    print(f'Servidor escuchando en {host}:{puerto}')

    # Esperar a que llegue una conexión
    conexion, direccion = servidor_socket.accept()
    print(f'Conexión aceptada de {direccion}')

    # Establecer el socket de conexión como bloqueante
    conexion.setblocking(True)

    # Recibir el mensaje
    mensaje = conexion.recv(1024).decode()

    # Mostrar el mensaje recibido
    print(f'Mensaje recibido: {mensaje}')

    # Cerrar la conexión y el socket del servidor
    conexion.close()
    servidor_socket.close()
 #Llama a la función para iniciar el servidor
servidor()
#while(1):
#    servidor()
```



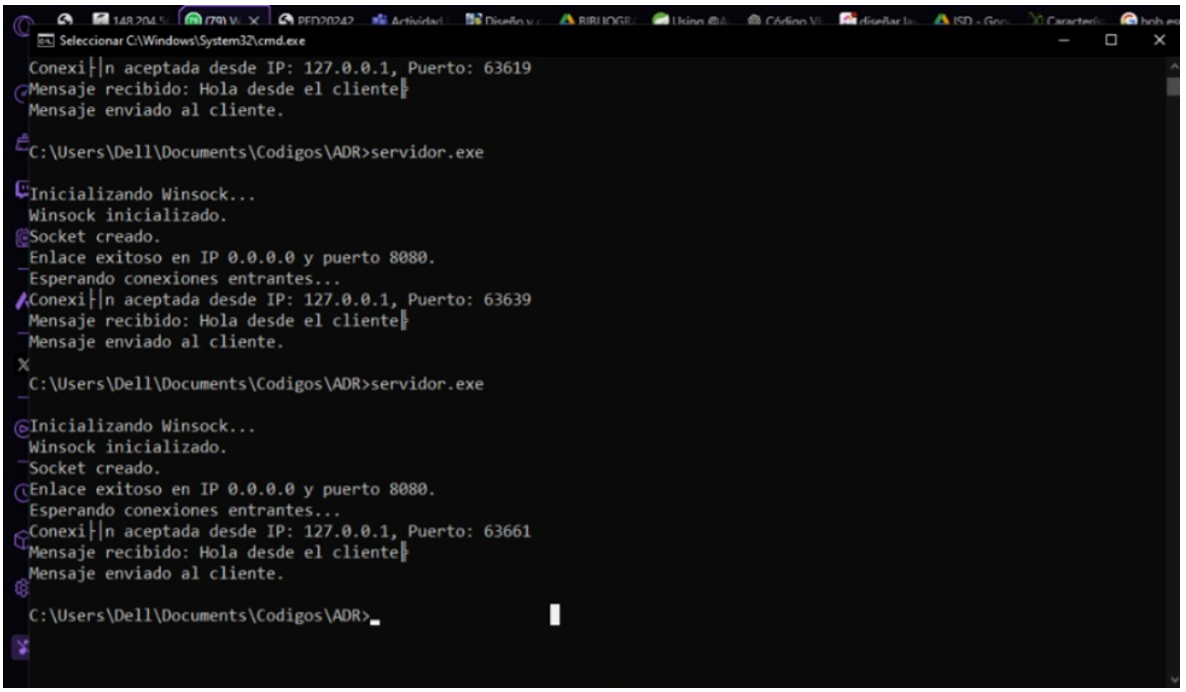
Cliente

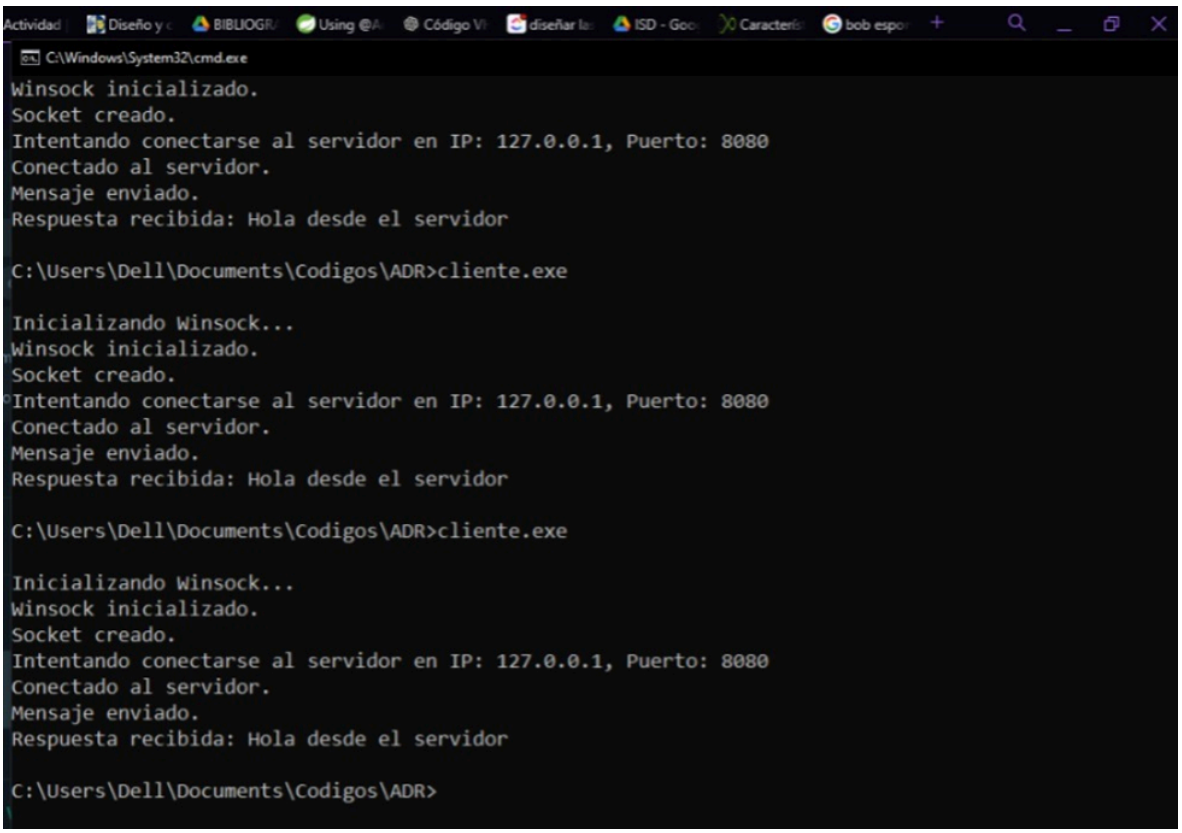
El cliente se configura para conectarse al servidor en la dirección IP 127.0.0.1 y el puerto 8080. Tras establecer la conexión, el cliente envía un mensaje al servidor y luego cierra la conexión.

El cliente también se ejecuta en modo bloqueante, por lo que la función `connect` detiene la ejecución del programa hasta que el servidor acepte la conexión.

Código del cliente:

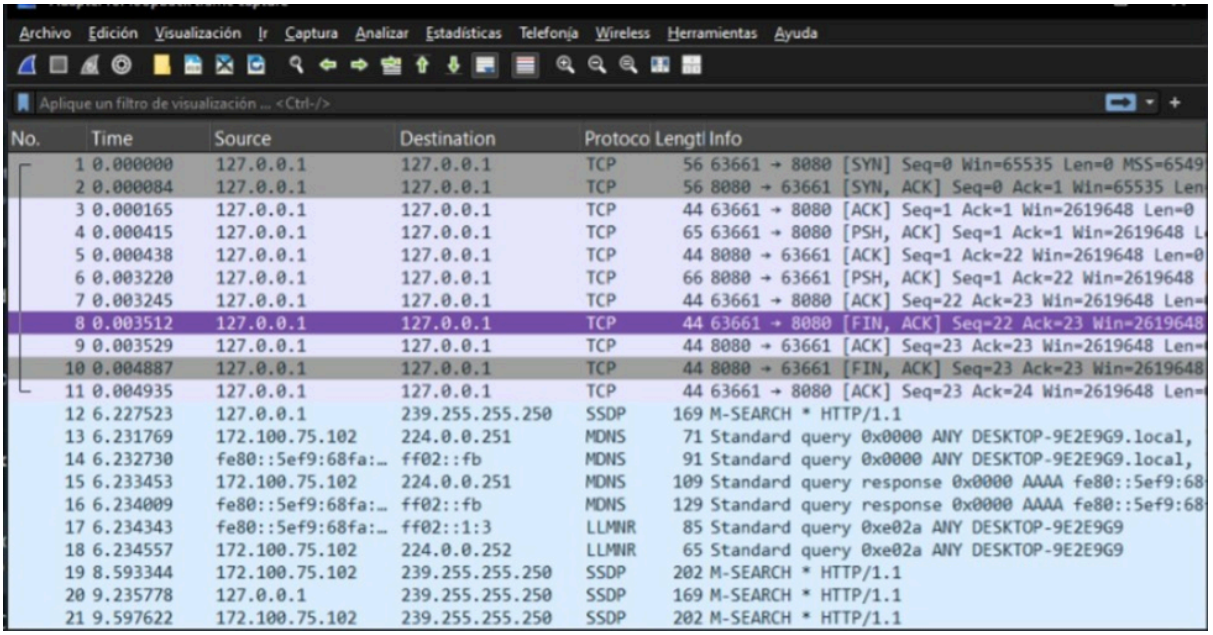
```
import socket
def cliente(mensaje, host='127.0.0.1', puerto=8080):
# Crear un socket TCP/IP
    cliente_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Establecer el socket como bloqueante
    cliente_socket.setblocking(True)
    # Conectar el socket al servidor
    cliente_socket.connect((host, puerto))
    print(f'Conectado a {host}:{puerto}')

    # Enviar el mensaje al servidor
    cliente_socket.sendall(mensaje.encode())

    # Cerrar el socket del cliente
    cliente_socket.close()
    print(f'Mensaje enviado al servidor: {mensaje}')

# Llama a la función con el mensaje que quieres enviar
cliente('Hola, servidor!')
```





Análisis de tráfico con Wireshark

Utilizando Wireshark, se capturaron los paquetes de datos intercambiados entre el cliente y el servidor. Durante la práctica, se observó que al iniciar la conexión, el protocolo TCP asegura un intercambio fiable de datos, mostrando las direcciones IP de origen y destino, además de la estructura de los paquetes enviados. Esto permitió verificar que el mensaje se transmitió de manera correcta y ordenada. Se analizó el comportamiento de las tramas generadas en la conexión, corroborando que el protocolo TCP cumple con su propósito de ofrecer una transmisión de datos confiable.



CONCLUSIONES

Navarrete Becerril Sharon Anette:

La implementación de servidores eficientes que gestionan miles de conexiones simultáneamente con mínima latencia es clave en industrias como los videojuegos en línea, plataformas financieras y el comercio electrónico, donde el desempeño del servidor influye directamente en la satisfacción y retención de usuarios. Esto ofrece la posibilidad de generar ingresos a través de la consultoría en optimización de redes o mediante el desarrollo de productos que aprovechen capacidades avanzadas de conectividad. Además, diseñar herramientas o complementos que integren sockets no bloqueantes en aplicaciones actuales, así como impartir cursos especializados en programación de redes, puede abrir oportunidades lucrativas y consolidar la reputación como experto en el campo

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.