Sockets orientados a conexiones bloqueantes

Un socket orientado a conexiones bloqueantes es una implementación de comunicación en red que sigue el modelo de cliente-servidor. En este modelo, las operaciones de red, como conectar, enviar y recibir datos, son bloqueantes. Esto significa que la ejecución del programa se detiene en estas operaciones hasta que se completen. Los sockets orientados a conexiones usan el protocolo TCP (Transmission Control Protocol), que garantiza una comunicación fiable y ordenada entre el cliente y el servidor. Antes de enviar cualquier dato, se establece una conexión entre ambos extremos.

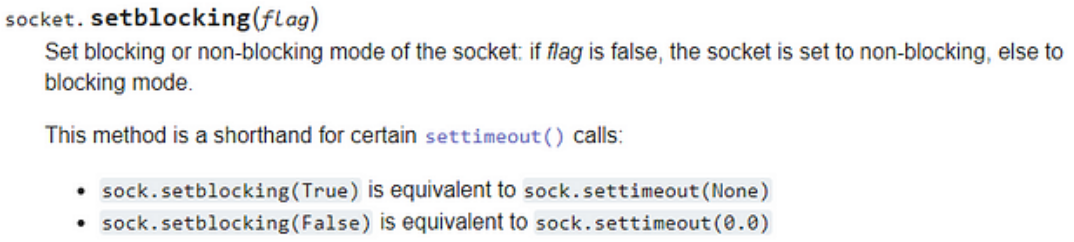
Estos sockets son útiles para situaciones donde se necesita una comunicación fiable y secuencial. Algunas de sus aplicaciones incluyen la transferencia de archivos, servicios de mensajería, servicios web y juegos en red. En el contexto de transferencia de archivos, un cliente puede enviar un archivo a un servidor de manera fiable y ordenada. En servicios de mensajería, las aplicaciones de chat pueden aprovechar los sockets bloqueantes para asegurar que los mensajes se envíen y reciban en el orden correcto. Los servicios web utilizan sockets TCP para gestionar las solicitudes y respuestas entre navegadores y servidores. En juegos en red, los sockets bloqueantes aseguran una comunicación sincronizada entre los jugadores.

Un par de características importantes son:

- **Orientado a conexiones:** Utiliza el protocolo TCP (Transmission Control Protocol), que garantiza una comunicación fiable y ordenada entre el cliente y el servidor. Antes de enviar cualquier dato, se establece una conexión entre ambos extremos.
- **Bloqueante:** Las llamadas a funciones como connect, send, recv y accept son bloqueantes. La ejecución del programa se detiene hasta que estas operaciones se completan.

Las aplicaciones de comunicación en tiempo real, como el chat en línea y los juegos multijugador, se benefician de la naturaleza fiable y ordenada de los sockets TCP. En aplicaciones distribuidas, como microservicios y computación distribuida, diferentes servicios o nodos en una red se comunican de manera fiable utilizando sockets bloqueantes. En microservicios, los diferentes componentes de una aplicación pueden comunicarse entre sí para proporcionar una funcionalidad completa. En computación distribuida, las tareas se distribuyen a través de múltiples nodos, y la comunicación confiable es esencial para coordinar estas tareas.

Hay bibliotecas y métodos para establecer el estado de un socket como bloqueante o no bloqueante en diversos lenguajes de programación, para el caso de la siguiente imagen está en el lenguaje de programación de Python:



`socket.` **`setblocking`**`(`*`flag`*`)`

Set blocking or non-blocking mode of the socket: if *flag* is false, the socket is set to non-blocking, else to blocking mode.

This method is a shorthand for certain `settimeout()` calls:

- `sock.setblocking(True)` is equivalent to `sock.settimeout(None)`
- `sock.setblocking(False)` is equivalent to `sock.settimeout(0.0)`

Desarrollo:

Inicialmente creamos un proyecto en Visual Studio Code, basta con crear una carpeta y dos scripts con extensión .py para esta práctica se nombraron de la siguiente manera:

- cliente.py
- servidor.py

Ahora se buscará en la presente práctica enviar un mensaje al servidor a través de un cliente, siguiendo el flujo que el servidor recibirá el mensaje si y solo si el servidor lo permite, para ello se deberá crear una conexión en caso de que la conexión falle el servidor se encargará de bloquear la conexión, el flujo en gran parte de esta práctica es debido a que

La función setblocking(flag) se utiliza para establecer el modo de operación de un socket, determinando si las operaciones sobre el socket serán bloqueantes o no bloqueantes.

- **Modo bloqueante:** Cuando flag es True, el socket está en modo bloqueante. En este modo, las operaciones de red como connect, send, recv y accept bloquearán la ejecución del programa hasta que se completen. Por ejemplo, una llamada a recv esperará hasta que haya datos disponibles para leer antes de devolver el control al programa.
- **Modo no bloqueante:** Cuando flag es False, el socket está en modo no bloqueante. En este modo, las operaciones de red no detendrán la ejecución del programa. Si no se puede completar una operación inmediatamente, la operación devolverá un error en lugar de bloquear la ejecución.

La función recv(bufsize) se utiliza para recibir datos desde un socket. bufsize es el número máximo de bytes que se leerán de una sola vez. Esta llamada es bloqueante por defecto, lo que significa que la ejecución del programa se detendrá hasta que lleguen datos al socket.

- **En modo bloqueante:** La llamada a recv esperará hasta que haya datos disponibles para leer. Si no hay datos disponibles, el programa se detendrá en esta línea hasta que lleguen los datos.
- **En modo no bloqueante:** La llamada a recv devolverá inmediatamente. Si no hay datos disponibles, lanzará una excepción BlockingIOError o similar, indicando que no hay datos para leer en este momento.

Los dos métodos anteriores mencionados van a poder hace que establezca una conexión pero que al momento en que dicha conexión presente una falla o alguna interferencia el socket bloqueante se encargará de realizar el bloqueo a la transferencia de datos.

Para comenzar el desarrollo de la presente práctica se va a tener que agregar el siguiente fragmento de código al archivo creado llamado cliente.py como se muestra a continuación:

```
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!')
```

La lógica anterior tiene la función de crear una función llamada cliente que se conecta a un servidor TCP en la dirección IP 127.0.0.1 y el puerto 8080. La función establece una conexión con el servidor, envía un mensaje y luego cierra la conexión. Finalmente, imprime mensajes para confirmar la conexión y el envío del mensaje.

Ahora se procede a agregar la lógica para el archivo servidor.py como el que se muestra a continuación:

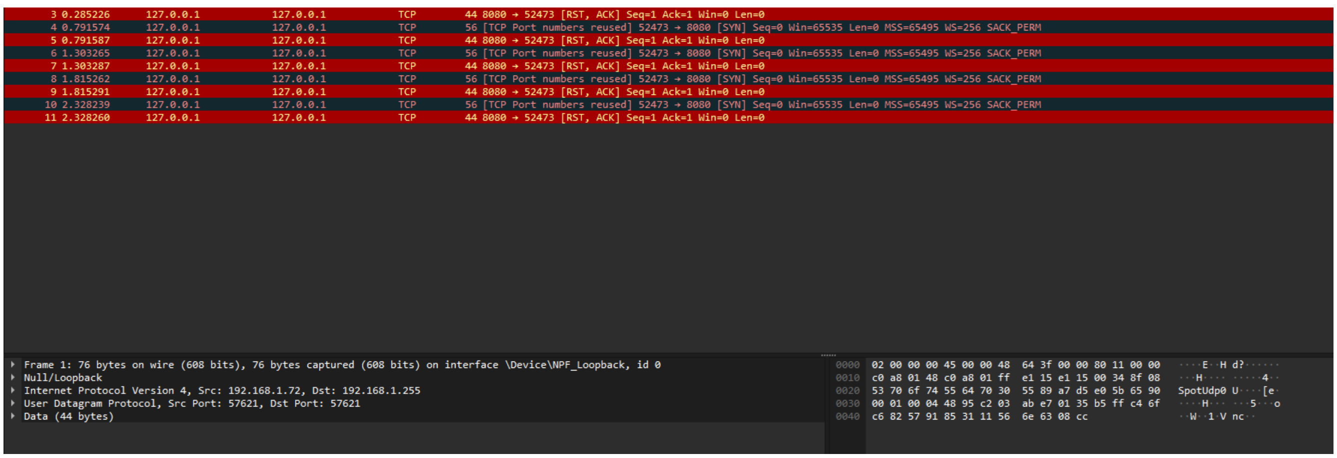
```
import socket

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()
```

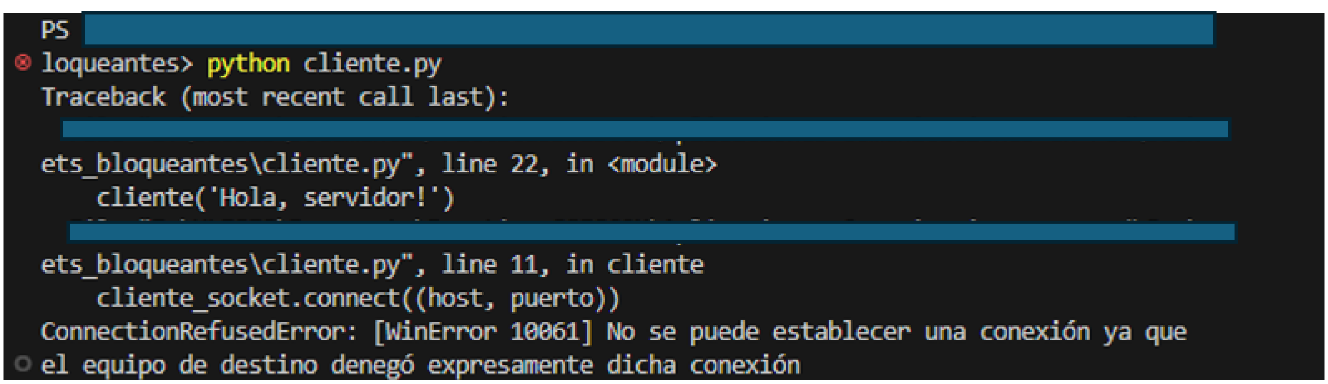
```python
# Llama a la función para iniciar el servidor
servidor()
```

El código define una función servidor que crea y configura un servidor TCP/IP en la dirección 127.0.0.1 y puerto 8080. Primero, establece un socket bloqueante y lo vincula al puerto especificado. Luego, el servidor escucha conexiones entrantes, acepta una conexión y recibe un mensaje de hasta 1024 bytes, el cual es decodificado y mostrado. Finalmente, cierra la conexión y el socket del servidor.

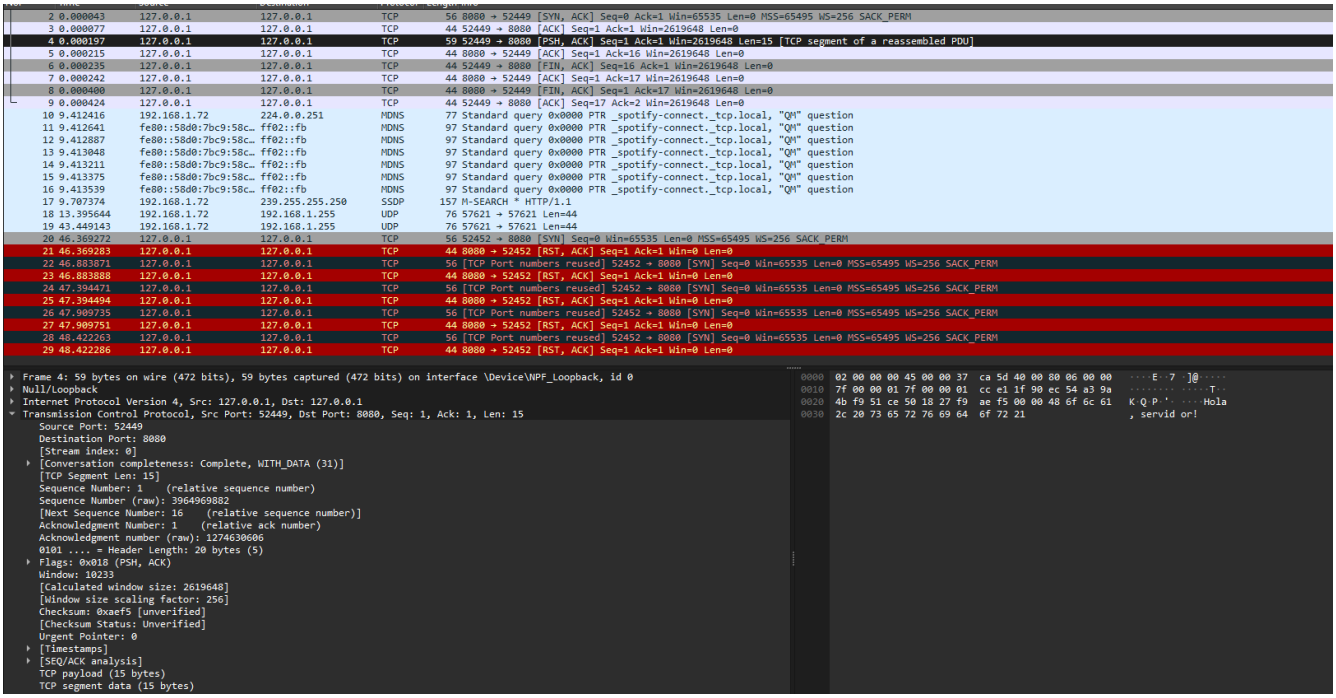
Una vez comprendido y desarrollada la lógica en nuestro IDE procedemos a la ejecución:

Con el apoyo de la herramienta de Wireshark que se instaló en la práctica anterior veremos el comportamiento del traspaso de los paquetes de datos a través del protocolo TCP, si intentamos ejecutar el cliente antes que el servidor habrá un bloqueo.





Ahora si se intenta establecer conexión una vez que el servidor se encuentre en ejecución, se procederá a mandar el mensaje a través del protocolo TCP como se muestra a continuación:




```
python servidor.py
Servidor escuchando en 127.0.0.1:8080
Conexión aceptada de ('127.0.0.1', 52482)
Mensaje recibido: Hola, servidor!
```

```
loqueantes> python cliente.py
Conectado a 127.0.0.1:8080
Mensaje enviado al servidor: Hola, servidor!
```