



Instituto Politécnico Nacional

ESCOM

“Escuela Superior de Cómputo”



Errores con programas no bloqueantes

Alumnos:

Pimentel González Ricardo Antonio.

Mota Domínguez Jesús Geovanni.

Grupo: 6CV1.

21/03/2025

***Aplicaciones para comunicaciones
en red.***

The image shows two windows. The top window is a Windows command prompt with the following commands and output:

```
C:\Users\jesus>cd Desktop
C:\Users\jesus\Desktop>cd PracticasRedes2
C:\Users\jesus\Desktop\PracticasRedes2>cd Tareas
C:\Users\jesus\Desktop\PracticasRedes2\Tareas>py server.py
[INICIO] Servidor escuchando en 192.168.100.9:12345
[NUEVA CONEXIÓN] Cliente conectado desde ('192.168.100.81', 57248)
[ACTIVO] Conexiones activas: 1
[DESCONECTADO] Cliente ('192.168.100.81', 57248) se ha desconectado
[NUEVA CONEXIÓN] Cliente conectado desde ('192.168.100.81', 50862)
[ACTIVO] Conexiones activas: 1
[DESCONECTADO] Cliente ('192.168.100.81', 50862) se ha desconectado
```

The bottom window is a VirtualBox terminal running Ubuntu. It shows a ping command and the execution of a client.py script:

```
geovanni@geovanni-VirtualBox: ~/Escritorio/Redes2JGMD/21032025
geovanni@geovanni-VirtualBox:~$ ping 192.168.100.9
PING 192.168.100.9 (192.168.100.9) 56(84) bytes of data:
64 bytes from 192.168.100.9: icmp_seq=1 ttl=128 time=0.907 ms
64 bytes from 192.168.100.9: icmp_seq=2 ttl=128 time=1.01 ms
^Z
[1]+  Detenido                  ping 192.168.100.9
geovanni@geovanni-VirtualBox:~$ cd Escritorio/Redes2JGMD/21032025/
geovanni@geovanni-VirtualBox:~/Escritorio/Redes2JGMD/21032025$ python3 client.py
geovanni@geovanni-VirtualBox:~/Escritorio/Redes2JGMD/21032025$ python3 client.py
geovanni@geovanni-VirtualBox:~/Escritorio/Redes2JGMD/21032025$
```

Al poner los programas con sockets no bloqueantes lo que podemos observar es que el servidor escucha y el cliente se conecta, sin embargo en el servidor si bien recibe la conexión se mantiene activo pero el cliente no encuentra respuesta del servidor y por lo tanto lo desconecta.

The image shows a code editor with two files: server.py and client.py. The server.py code is as follows:

```
10
11 def handle_client(conn, addr):
12     print(f"[NUEVA CONEXIÓN] Cliente conectado desde {addr}")
13     conn.setblocking(False) # Hacer el socket no bloqueante
14
15     try:
16         while True:
17             ready_to_read, _, _ = select.select([conn], [], [])
18             if conn in ready_to_read:
19                 data = conn.recv(1024).decode('utf-8')
20                 if not data:
21                     break
22                 print(f"[RECIBIDO] {addr}: {data}")
23                 conn.sendall(f"Echo: {data}".encode('utf-8'))
24     except ConnectionResetError:
25         print(f"[ERROR] Conexión con {addr} reseteada")
26     finally:
27         conn.close()
28     print(f"[DESCONECTADO] Cliente {addr} se ha desconectado")
29
30 # Crear socket del servidor
31 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
32 server.bind((HOST, PORT))
33 server.listen(MAX_CLIENTS)
34 server.setblocking(False) # Hacer el servidor no bloqueante
35 print(f"[INICIO] Servidor escuchando en {HOST}:{PORT}")
36
37 while True:
```

The client.py code is as follows:

```
def client_workload():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.setblocking(False) # Hacer el socket no bloqueante

    try:
        client.connect((HOST, PORT))
    except BlockingIOError:
        pass # Conexión en proceso, no bloquea

    for i in range(5):
        message = f"Mensaje {i+1} desde el cliente"
        # Verifica si el socket está listo para escribir
        ready_to_write, _, _ = select.select([], [client], [], 1)

        if client in ready_to_write:
            client.sendall(message.encode('utf-8'))

        # Verifica si el socket tiene datos para leer
        ready_to_read, _, _ = select.select([client], [], [], 1)

        if client in ready_to_read:
```

Asi los hacemos no bloqueante, a continuación las fallas que pudimos percatarnos y que posiblemente saldrían.

The screenshot shows a VS Code editor with a file named `Servidor5.py` open. The code is a Python script for a simple server. It imports `socket`, `threading`, and `time`. It sets `HOST` to `'10.0.2.15'` and `PORT` to `12345`. The `handle_client` function receives a connection and address, prints a message, sets the socket to non-blocking, and enters a loop to receive data. It prints the received data and sends it back. The `main` function binds the server to the host and port and starts a thread to handle clients.

```
1 import socket
2 import threading
3 import time
4
5 # Configuración del servidor
6 HOST = '10.0.2.15' # Dirección local
7 PORT = 12345      # Puerto de escucha
8 MAX_CLIENTS = 5
9
10 def handle_client(conn, addr):
11     print(f"[NUEVA CONEXION] Cliente conectado desde {addr}")
12     conn.setblocking(False) # Hacer el socket del cliente no bloqueante
13     try:
14         while True:
15             try:
16                 data = conn.recv(1024).decode('utf-8')
17                 if not data:
18                     break
19                 print(f"[RECIBIDO] {addr}: {data}")
20                 conn.sendall(f"Echo: {data}".encode('utf-8'))
21             except BlockingIOError:
22                 time.sleep(0.01) # Espera un poco para evitar consumir CPU
23             except Exception as e:
24                 print(f"[ERROR] Cliente {addr}: {e}")
25
26 if __name__ == '__main__':
27     server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
28     server.bind((HOST, PORT))
29     server.listen(5)
30     print(f"Escuchando conexiones en {HOST}:{PORT}")
31     while True:
32         conn, addr = server.accept()
33         thread = threading.Thread(target=handle_client, args=(conn, addr))
34         thread.start()
35         print(f"Nuevo cliente {addr} conectado")
36     server.close()
```

The terminal output shows the execution of the script. It displays the server binding to the address and port, and then shows the error `OSError: [Errno 98] Address already in use` when the script is run again, indicating that the port is still occupied.

Cliente

- `client.connect((HOST, PORT))`
Puede fallar con `BlockingIOError` si la conexión no se completa de inmediato.
- `client.recv(1024)`
Puede lanzar `BlockingIOError` si no hay datos disponibles.

Servidor

- `server.accept()`
Puede fallar con `BlockingIOError` si no hay clientes conectándose.
- `conn.recv(1024)`
Puede lanzar `BlockingIOError` si no hay datos en el buffer.

General

- `select.select()`
Puede generar pequeños retrasos si ningún socket está listo.
- Sin un `timeout` en `select.select()` el programa puede quedarse esperando indefinidamente.

En nuestro caso el programa servidor quedo esperando indefinidamente mientras que el cliente cerraba la conexión puesto que no lograba conectarse a su parecer.