



Nombre: Luis Alberto Vargas González.

Fecha: 24/01/2025.

U2 A1: Socket TCP.

Clase: Sistemas Distribuidos.

Maestría en Ciberseguridad.

Descripción de programas

En esta práctica se construye un socket TCP el cual fundamentalmente trabaja con conexiones preestablecidas a diferencia de UDP que no lo hace, por lo tanto, el código fuente escrito nuevamente en lenguaje de programación Python es muy similar al de UDP con unas ligeras diferencias en la construcción del socket, en donde se definen el objeto creado en el programa, junto con los argumentos de la función.

(socket.AF_INET, socket.SOCK_STREAM), en donde definimos previamente la dirección de red ipv4 en este caso igual a UDP 127.0.0.1, y el argumento SOCK_Stream es una constante que indica que el socket utilizará el protocolo TCP (Transmission Control Protocol).

Código de cliente TCP

```
import socket

def iniciar_cliente_tcp():
    HOST = '127.0.0.1' # Dirección IP del servidor
    PORT = 56432 # Puerto del servidor

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((HOST, PORT)) # Conectarse al servidor
        print("Conectado al servidor. Escribe 'exit' para terminar la conexión.")

    while True:
        mensaje = input("Escribe tu mensaje: ") # Leer mensaje desde la consola
```

```

if mensaje.lower() == "exit":
    print("Cerrando la conexión...")
    client_socket.sendall(mensaje.encode('utf-8')) # Enviar 'exit' al servidor
    break
    client_socket.sendall(mensaje.encode('utf-8')) # Enviar mensaje
    data = client_socket.recv(1024) # Recibir respuesta
    print(f'Respuesta del servidor: {data.decode('utf-8')}')

if __name__ == "__main__":
    iniciar_cliente_tcp()

```

Código de server TCP.

```

import socket

def iniciar_servidor_tcp():
    HOST = '127.0.0.1' # Dirección IP local
    PORT = 56432 # Puerto para escuchar

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.bind((HOST, PORT)) # Asociar IP y puerto
        server_socket.listen() # Escuchar conexiones entrantes
        print(f'Servidor TCP escuchando en {HOST}:{PORT}')

        conn, addr = server_socket.accept() # Aceptar una conexión
        print(f'Conexión establecida con {addr}')
        with conn:
            while True:
                data = conn.recv(1024) # Recibir datos

```

```
if not data:
    print("Cliente desconectado.")
    break
print(f'Mensaje recibido: {data.decode('utf-8')}')
# Enviar un eco de vuelta al cliente
conn.sendall(data)

if __name__ == "__main__":
    iniciar_servidor_tcp()
```

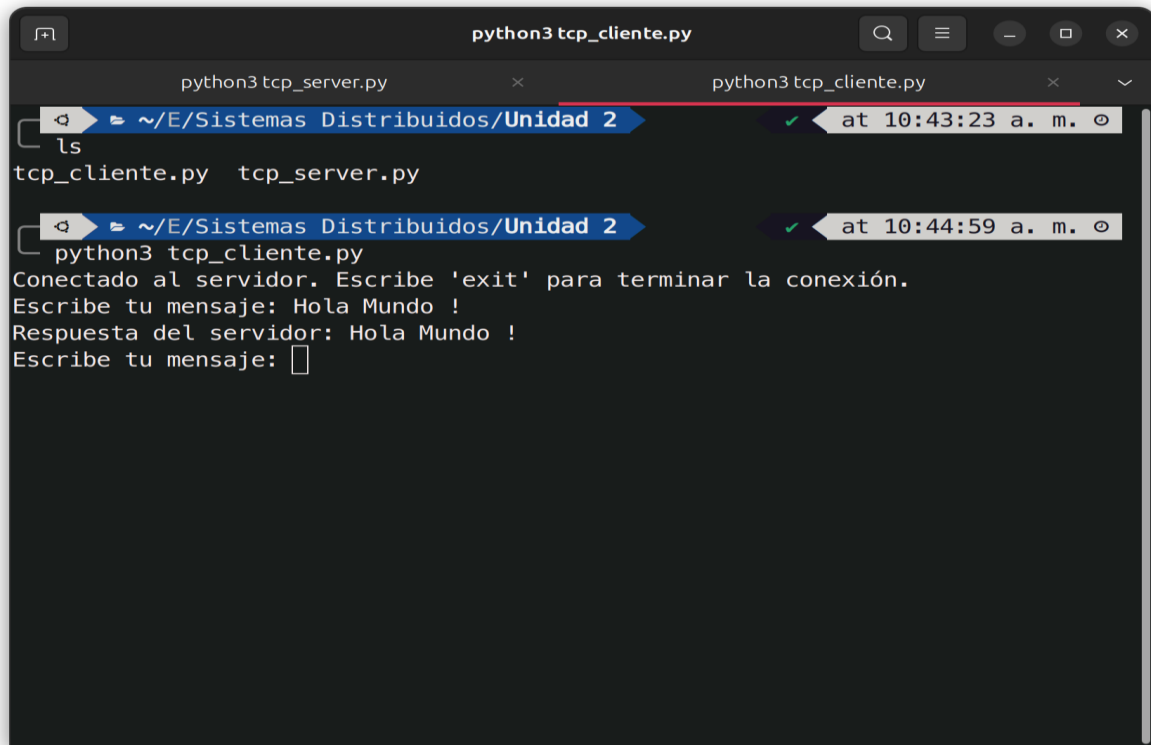
Además, hay otra diferencia ligera: en UDP usamos el método `sendto()` para enviar los datos sin establecer una conexión y en TCP utilizamos `sendall()` el cual asegura que una vez establecida la conexión se envíen todos los datos del usuario.

Para establecer las conexiones entre cliente y server, se utiliza un handshake para establecer la comunicación entre server y cliente de manera fiable, esto no se ve reflejado directamente en el código ya que se realiza de manera interna al usar `connect()` y `accept()` en el servidor y cliente respectivamente.

Evidencia de funcionamiento.

A terminal window titled "python3 tcp_server.py" with two tabs: "python3 tcp_server.py" and "python3 tcp_cliente.py". The window shows the execution of a TCP server and client. The first command is "ls" which lists "tcp_cliente.py" and "tcp_server.py". The second command is "python3 tcp_server.py", which outputs "Servidor TCP escuchando en 127.0.0.1:56432", "Conexión establecida con ('127.0.0.1', 35814)", and "Mensaje recibido: Hola Mundo !".

```
python3 tcp_server.py
python3 tcp_cliente.py
~/E/Sistemas Distribuidos/Unidad 2 at 10:43:02 a. m.
ls
tcp_cliente.py tcp_server.py
~/E/Sistemas Distribuidos/Unidad 2 at 10:44:47 a. m.
python3 tcp_server.py
Servidor TCP escuchando en 127.0.0.1:56432
Conexión establecida con ('127.0.0.1', 35814)
Mensaje recibido: Hola Mundo !
```



The screenshot shows a terminal window with two tabs: 'python3 tcp_server.py' and 'python3 tcp_cliente.py'. The active tab is 'python3 tcp_cliente.py'. The terminal output shows the following sequence of events:

```
python3 tcp_cliente.py
ls
tcp_cliente.py  tcp_server.py

python3 tcp_cliente.py
Conectado al servidor. Escribe 'exit' para terminar la conexión.
Escribe tu mensaje: Hola Mundo !
Respuesta del servidor: Hola Mundo !
Escribe tu mensaje: 
```

Como se puede notar al igual que en el programa de udp primero se levanta el servidor después el cliente, cuando el cliente envía el mensaje el servidor lo recibe, lo imprime en pantalla y devuelve el mensaje recibido para su confirmación de recepción.

Conclusiones

Al finalizar esta actividad nos pudimos percatar que la realización de la misma conlleva un esfuerzo pequeño pues el algoritmo en si es muy sencillo de implementar por lo tanto es fácil de entender y de modificar en caso de ser requerido. Es importante entender de que va el protocolo UDP y TCP para poder entender a mayor profundidad los sistemas distribuidos, pues tanto TCP como UDP son usados para diferentes tipos de sistemas y en distintas topologías de red.