

### ***Actividad***

Realiza un programa en el lenguaje de tu preferencia, que de manera paralela (programación paralela) realice el envío, recepción y almacenamiento de mensajes entre tus "n" nodos virtuales a través de sockets, con las siguientes características:

- Desde cualquier nodo se debe poder enviar un mensaje escrito por el usuario.
- Todos los mensajes deben de responderse automáticamente de recibido.
- Los mensajes se deben de almacenarse en ambos nodos involucrados.
- Todo mensaje debe de incluir el tiempo del reloj del nodo que envía el mensaje.

### ***Investigación para solucionar actividad***

¿Qué es socket?

Un socket es una abstracción de software que se utiliza para establecer una comunicación bidireccional entre dos programas o dispositivos a través de una red, ya sea en una misma máquina o entre máquinas distintas. Los sockets permiten que los programas se comuniquen enviando y recibiendo datos, lo que es fundamental para la comunicación en redes de computadoras.

¿Qué es programación paralela?

Es una técnica de programación en la que varias tareas o procesos se ejecutan simultáneamente en paralelo, con el objetivo de mejorar el rendimiento y la eficiencia de un sistema informático. En lugar de ejecutar una tarea tras otra de manera secuencial, la programación paralela nos permite que múltiples tareas se ejecuten al mismo tiempo, aprovechando así los recursos de hardware de manera más eficaz.

### ***Implementar lo solución***

Primero estableceremos que será el servidor, así que para programarlo necesitamos plantear las bibliotecas que serán necesarias

```
# Importa la biblioteca 'socket' para habilitar la comunicación de red,  
# lo que permite la creación y el uso de sockets para establecer conexiones
```

```
# y transmitir datos entre el cliente y el servidor.
import socket

# Importa la biblioteca 'threading' que proporciona herramientas para
trabajar
# con subprocesos, lo que permite manejar múltiples conexiones de clientes
de forma concurrente.
import threading

# Importa la biblioteca 'time' para trabajar con funciones relacionadas con
el tiempo,
# como la obtención de la marca de tiempo actual, que se utiliza en la
función de manejo
# de clientes para marcar los mensajes con la hora en que se enviaron.
import time
```

Posteriormente estableceremos una variable que adquirirá los valores por defecto de nuestro servidor, el cual se le entregará su IP del host y el tipo de protocolo que seguirá, posteriormente esperara alguna respuesta de algún “cliente”

```
# Crea un socket de tipo AF_INET (IPv4) y SOCK_STREAM (TCP)
socketServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Enlaza el socket al localhost en el puerto 8000
# En el caso para obtener este ip, lo conseguimos usando
# el comando ipconfig en la terminal de windows
socketServer.bind(('192.168.100.50', 8000))

# Escucha hasta 5 conexiones entrantes en el socket
socketServer.listen(5)
print("Servidor esperando conexiones...")
```

Se mantendrá un ciclo while, el cual estará esperando continuamente una conexión con algún cliente, para poder mantener “activos” a diferentes clientes, se asignará un hilo para poder procesar sus actividades.

```
while True:
    # Acepta una conexión entrante y obtiene el socket del cliente y su
    # dirección
    socketCliente, ipCliente = socketServer.accept()
    print(f"Nueva conexión establecida con {ipCliente}")

    # Inicia un subproceso para manejar al cliente
    manejandoCliente = threading.Thread(target=procesarCliente,
    args=(socketCliente, ipCliente))
    manejandoCliente.start()
```

Si accede correctamente, nuestro cliente tiene la posibilidad de interactuar con el servidor, el cual se mantendrá activo, hasta que finalice la conexión del cliente, lo único que estará procesando será el mensaje del cliente, sin embargo, se concatenará con otros datos, como lo serán, procesar el mensaje el cual le daremos un formato de tipo utf8, cuando el cliente, no envíe nada, el programa finalizaría, la actividad nos solicita que reportemos la hora en la que interactúa con nosotros, entonces le dimos un formato. Almacenamos en un diccionario, nuestros mensajes para posteriormente poder usarlo. Pero de mientras, cada que el cliente nos envía un mensaje, lo reflejaremos en nuestra “terminal” donde se esta ejecutando nuestro programa.

```
def procesarCliente(socketCliente, ipCliente):
    while True:
        try:
            # Recibe datos enviados por el cliente (hasta 1024 bytes)
            datos = socketCliente.recv(1024)

            # Si no se recibe ningún dato, se sale del bucle
            if not datos:
                break
```

```
        # Decodifica los datos recibidos en formato UTF-8 para obtener
el mensaje
        mensaje = datos.decode('utf-8')

        # Obtiene la marca de tiempo actual del servidor
        tiempoProcesado = time.strftime("%Y-%m-%d %H:%M:%S",
time.localtime())

        # Muestra el mensaje del cliente junto con la marca de tiempo y
su dirección
        print(f"Mensaje de {ipCliente} ({tiempoProcesado}): {mensaje}")

        # Almacena el mensaje en el diccionario de mensajes
        if ipCliente not in mensajes:
            mensajes[ipCliente] = []
        mensajes[ipCliente].append((tiempoProcesado, mensaje))

        # Envía una respuesta al cliente confirmando la recepción del
mensaje
        respuesta = f"Mensaje recibido de {ipCliente}"
        socketCliente.send(respuesta.encode('utf-8'))

    except Exception as error:
        # Si ocurre un error, se muestra en la consola y se sale del
bucle

        print(f"Ocurrió un error: {error}")
        break

    # Cierra la conexión con el cliente una vez que se completa la
comunicación
    socketCliente.close()
```

Cuando terminamos la programación de nuestro servidor, no tiene ningún funcionamiento, ya que nadie establece conexión con nosotros, entonces se necesitó diseñar un cliente, para

esto se necesita crear un nuevo archivo Python, entonces necesitamos plantear nuevamente las bibliotecas necesarias, las cuales son los siguientes;

```
# Importamos la biblioteca 'socket' para habilitar la comunicación de red,  
# el cual nos permite la creación y el uso de sockets para la comunicación  
entre  
# el cliente y el servidor.  
import socket  
  
# Importamos la biblioteca 'time' para trabajar con funciones relacionadas  
con el tiempo,  
# en este caso, se utiliza para obtener la marca de tiempo actual y  
formatearla.  
import time
```

Para poder comenzar, necesitamos establecer nuevamente una variable que adquiera las características para establecer conexión con la red, para esto ya no necesitamos usar un “bind”, entonces ahora solo necesitamos hacer una conexión a nuestro servidor, el cual solo debemos agregar la IP y el puerto que está expuesto del servidor para realizar una comunicación mutua.

```
# Crea un nuevo socket del cliente  
miSocket = socket.socket()  
  
# Establece una conexión con el servidor en localhost y el puerto 8000  
miSocket.connect(('192.168.100.50', 8000))
```

Una vez conectado, podemos mandar únicamente mensajes, que únicamente leerá nuestro servidor, entonces procedemos a enviar el mensaje, si se logra, recibimos una respuesta del servidor el cual nos indicará su IP, si en algún momento se interrumpe la conexión se estableció un except en cual, solo nos informará que hubo un error, y termina nuestra conexión con el servidor.

```
# Crea un nuevo socket del cliente
miSocket = socket.socket()

# Establece una conexión con el servidor en localhost y el puerto 8000
miSocket.connect(('192.168.100.50', 8000))

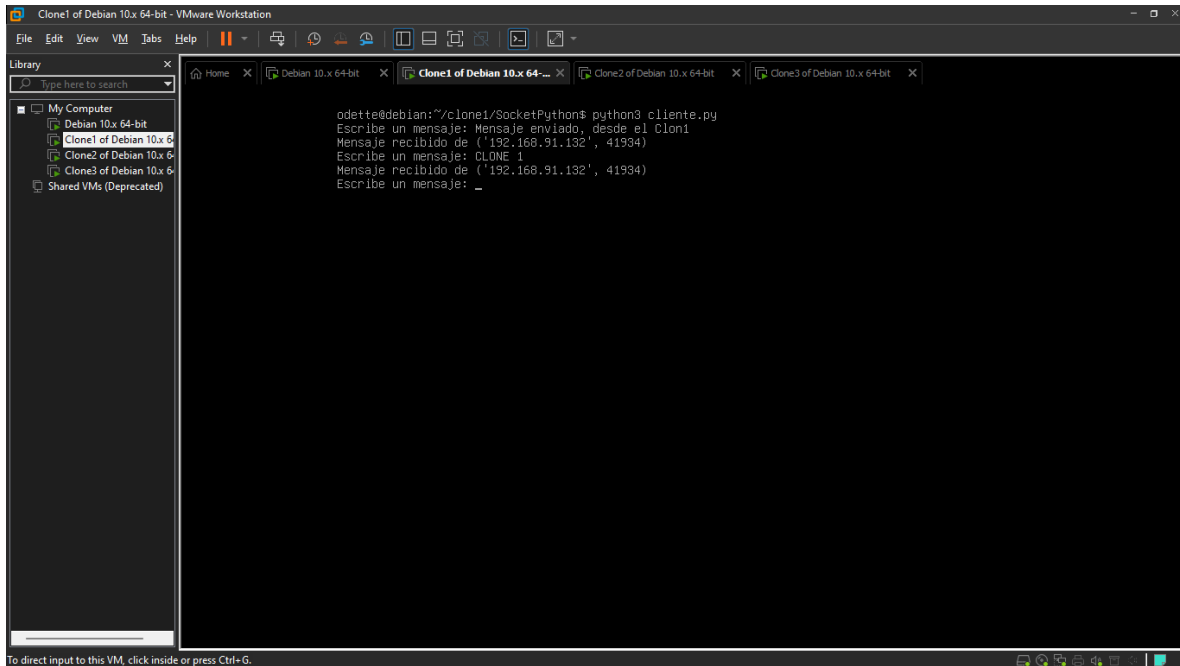
while True:
    # Obtiene un mensaje ingresado por el usuario
    mensaje = input("Escribe un mensaje: ")
    #Se agrego un try - except, ya que se considera el caso en el que el
servidor se desconecta a mitad
    #del proceso, así que es mejor tambien finalizar la actividad del
cliente con el servidor.
    try:
        # Llama a la función 'enviar_mensaje' para enviar el mensaje al
servidor
        enviar_mensaje(miSocket, mensaje)

        # Recibe una respuesta del servidor (hasta 1024 bytes) y la
decodifica
        respuesta = miSocket.recv(1024)
        print(respuesta.decode('utf-8'))
        #Cuando se interrumpe la comunicación con el servidor, se finaliza
la petición de más mensajes
        #y esto provoca que el programa al igual finalice de este lado.
    except ConnectionResetError as error:
        print("Se interrumpio la conexión con el servidor")
        break
```

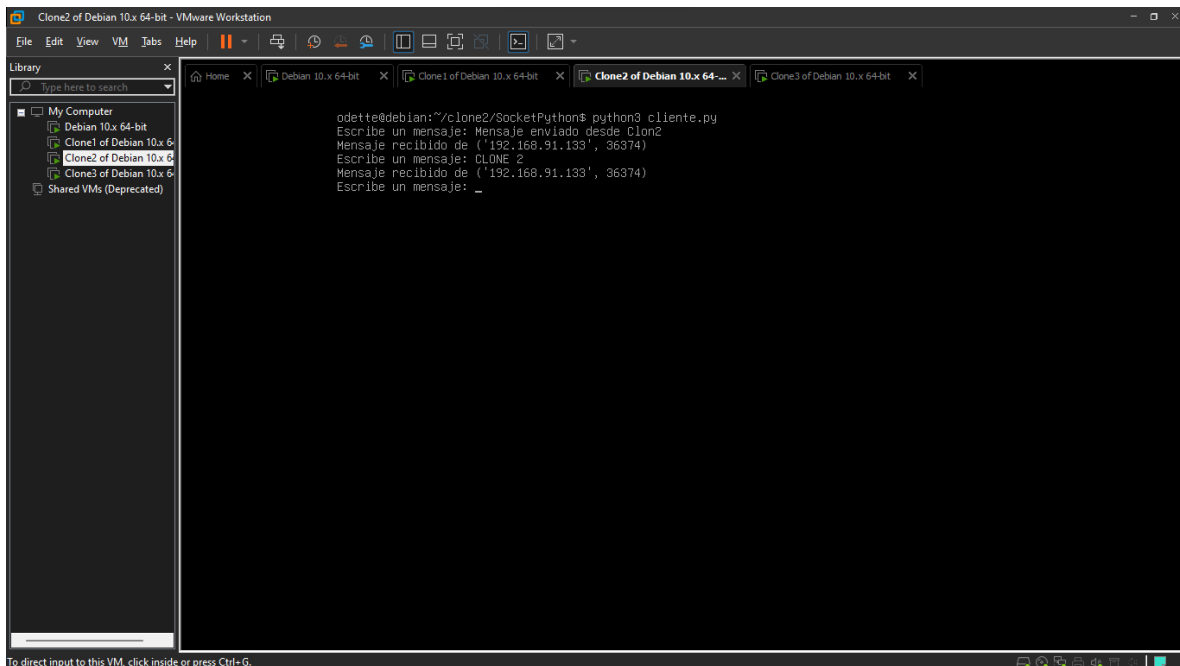
## Pruebas

A continuación, se mostrará la ejecución de los clientes, los cuales están interactuando directamente con el servidor, en cada cliente se muestra como se manda un mensaje distinto en cada cliente, para reconocerlos, los mensajes eran referente con cada cliente, como vemos

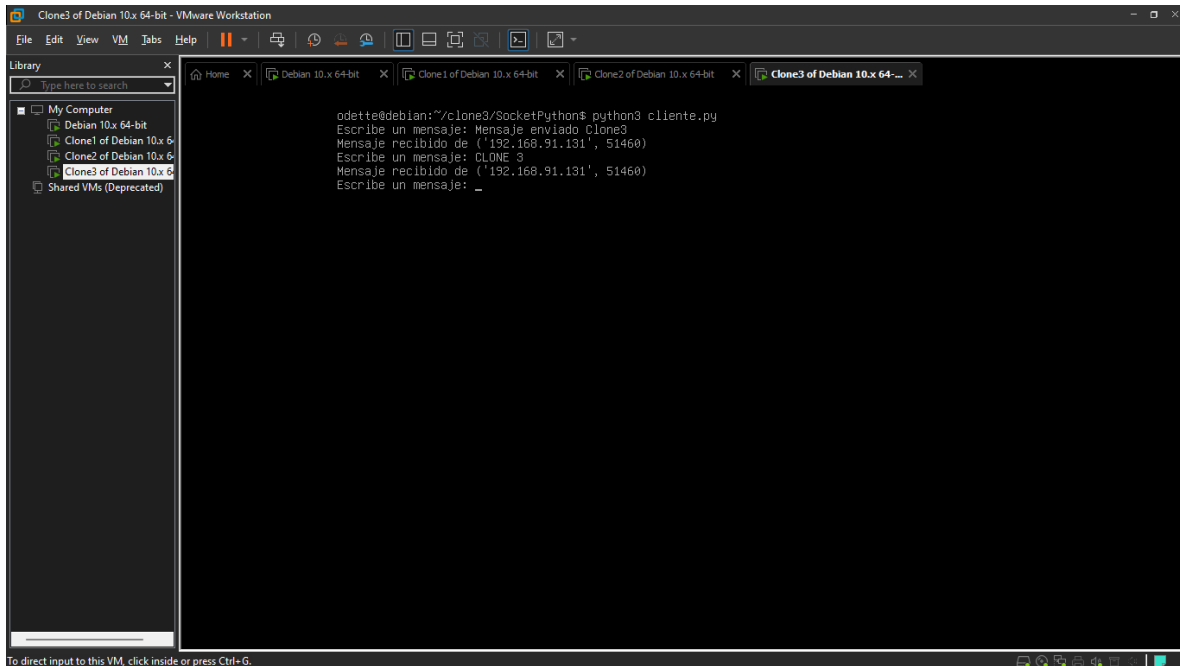
aquí recibimos respuesta de nuestro servidor, ¿Cómo sabemos que es el servidor?, porque previamente se vio cual era la IP, esto se hizo con ip address, para configurar el servidor con su propia IP y los clientes se conecten a esta misma IP.



```
odette@debian:~/clone1/SocketPython$ python3 cliente.py
Escribe un mensaje: Mensaje enviado, desde el Clon1
Mensaje recibido de ('192.168.91.132', 41934)
Escribe un mensaje: CLONE 1
Mensaje recibido de ('192.168.91.132', 41934)
Escribe un mensaje: _
```

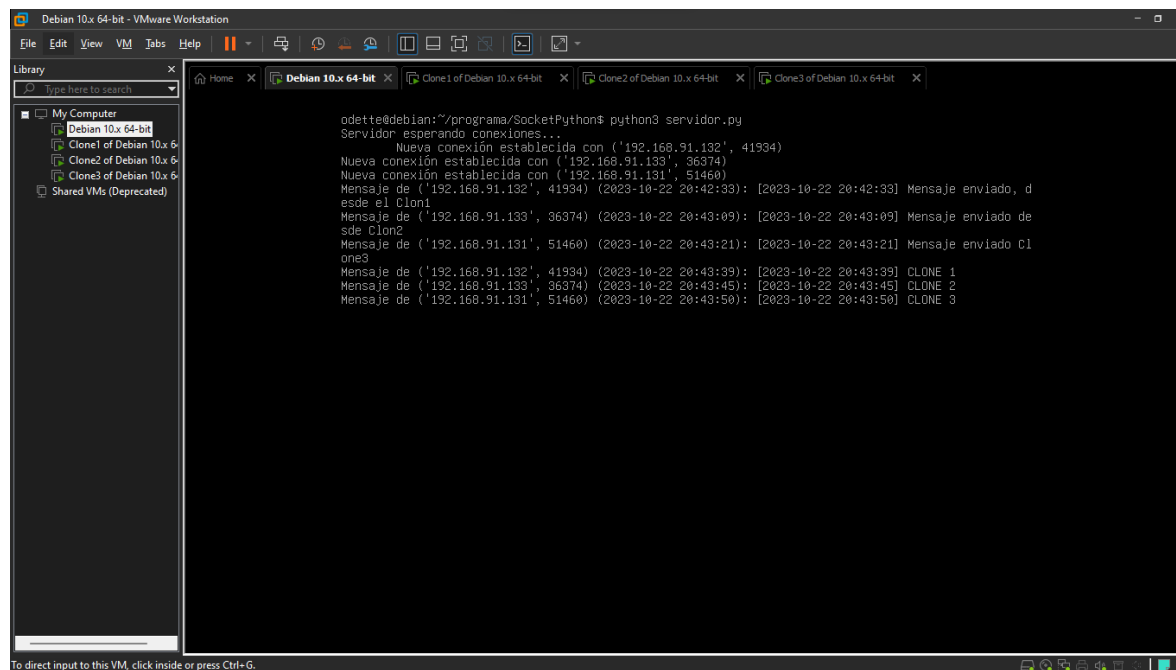


```
odette@debian:~/clone2/SocketPython$ python3 cliente.py
Escribe un mensaje: Mensaje enviado desde clon2
Mensaje recibido de ('192.168.91.133', 36374)
Escribe un mensaje: CLONE 2
Mensaje recibido de ('192.168.91.133', 36374)
Escribe un mensaje: _
```



En una máquina virtual, lo asignamos como el servidor, entonces este al recibir la conexión de algún cliente, nos lo expondrá en la pantalla, es importante destacar (al menos en Windows me ocurrió), hay que tener algún tipo de permiso por parte del Firewall para poder hacer este tipo de conexión, hay que permitirlo en redes privadas. Pero como vemos en la imagen, se muestra como recibe los mensajes de cada cliente, este mismo muestra la IP de cada uno de ellos, los cuales muestra además la fecha y hora que se realizó este mensaje





El consumo del programa con varias maquinas virtuales, no resulta ser tan demandante

Nombre	Estado	CPU	Memoria
<b>Aplicaciones (5)</b>			
Administrador de tareas		1.5%	64.7 MB
Brave Browser (17)		1.2%	1,690.1 MB
Sticky Notes (2)		0%	47.8 MB
VMware Workstation (32 bits) (3)		0%	61.9 MB
WhatsApp (2)		0%	247.9 MB

## Conclusiones

Este tipo de programa nos abre una posibilidad de como realizar trabajo a través de un servidor, siendo un cliente. Podríamos aprovechar los recursos, como lo haría SSH. Creo que este tipo de programación nos ofrece unas cuantas ventajas como lo son:

- **Entender la comunicación en red:** Aprender sobre sockets nos brinda una comprensión sólida de cómo funcionan las comunicaciones en red, un concepto fundamental en programación. Esto es útil no solo para la programación, sino también para comprender cómo funcionan Internet y las aplicaciones en línea.
- **Mejora de la eficiencia y el rendimiento:** Al utilizar sockets para la comunicación en red, podemos diseñar aplicaciones que sean más eficientes en términos de uso de recursos y más rápidas en la transmisión de datos. Esto es especialmente importante en aplicaciones de alto rendimiento.
- **Desarrollo de aplicaciones distribuidas:** podemos crear aplicaciones distribuidas que se ejecuten en varias máquinas y se comuniquen entre sí. Esto es fundamental en la construcción de sistemas distribuidos y en la computación en la nube.
- **Aplicaciones en el mundo real:** Muchas aplicaciones del mundo real utilizan sockets, por lo que aprender este tipo habilidad puede ser útil, desde el desarrollo de software hasta la administración de redes.

### *Referencias*

- Canal *codigofacilito*. (12 de marzo de 2018). Crear servidor sockets con Python - Bytes. YouTube. [https://youtu.be/nJYp3\\_X\\_p6c?si=ngnyDLXV13fIT1pG](https://youtu.be/nJYp3_X_p6c?si=ngnyDLXV13fIT1pG)
- Canal *codigofacilito*. (6 de febrero de 2018). Hilos en Python - Bytes. YouTube. <https://youtu.be/J9wOU5uWrjw?si=cp2L6pb1tQsSdpEe>
- Canal *Luis Munoz*. (25 de octubre de 2021). Sockets con Python - Hola Mundo. YouTube. [https://youtu.be/AU7nNGTqKT0?si=yEe\\_H8AFTsBHHk9](https://youtu.be/AU7nNGTqKT0?si=yEe_H8AFTsBHHk9)
- Canal *Luis Munoz*. (25 de octubre de 2021). Sockets con Python - Introducción. YouTube. <https://youtu.be/8gm1GcJR7R8?si=AOdwF82jt99oTLYJ>
- De La Cruz. C. (2023). Socket Python. Github. <https://github.com/CarlosOdetteDLCL080301/SocketPython>