CORPORACION UNIVERSITARIA IBEROAMERICANA
FACULTAD DE INGENIERIA
INGENIERIA DE SOFTWARE

ACTIVIDAD 3 - ARQUITECTURA CLIENTE SERVIDOR

AUTOR
DUBAN LEONARNO GUZMAN MEDINA
ID: 100126620

DOCENTE

JOAQUIN SANCHEZ

MAJAGUAL(SUCRE) – COLOMBIA

2024

Introducción:

La arquitectura de cliente-servidor es un modelo de diseño de software que permite la interacción entre dos entidades distintas: un cliente que solicita servicios de manera dinámica y un servidor que responde proporcionándolos. Este paradigma de red resulta fundamental para el desarrollo de aplicaciones distribuidas y las comunicaciones a través de internet, habilitando que sistemas diferentes interactúen de forma eficiente. En este contexto, Python se ha consolidado como uno de los lenguajes de programación más populares a la hora de construir aplicaciones en red, gracias a su sencillez sintáctica y su potente biblioteca estándar. Al utilizar el módulo socket, Python facilita enormemente la creación de software capaz de comunicarse a través de redes, ya sea una interna o a través de internet, estableciendo conexiones y transfiriendo datos de un modo flexible y dinámico.

Descripción del producto.

El sistema consiste en una sencilla arquitectura cliente-servidor construida en Python utilizando sockets. A continuación, se destacan sus características principales:

Servidor:

Escucha constante: El servidor está diseñado para admitir múltiples accesos simultáneos de clientes usando un modelo de escucha persistente.

Comunicación bidireccional: Permite el intercambio de mensajes entre cliente y servidor, garantizando que las solicitudes se procesen adecuadamente.

Manejo de conexiones: Cada conexión de un cliente se gestiona de manera independiente, lo que permite que el servidor continúe operando al interactuar con varios usuarios.

Cliente:

Simple enlace: El cliente se vincula fácilmente al servidor y envía textos para comunicarse.

Respuestas recibidas: Después de enviar un mensaje, el cliente puede obtener una réplica del servidor confirmando la recepción de su petición.

Código servidor:

```
import socket

# Crear el socket del servidor
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# El servidor se enlaza a una IP y un puerto
```

```
host = '127.0.0.1'  # Dirección de loopback para pruebas locales
port = 12345        # Puerto del servidor

server_socket.bind((host, port))

# Configurar el servidor para que escuche conexiones entrantes
server_socket.listen(5)  # Permite hasta 5 conexiones

print(f"Servidor escuchando en {host}:{port}...")

while True:
    # Acepta una conexión
    client_socket, addr = server_socket.accept()
    print(f"Conexión establecida desde {addr}")

    # Recibe datos del cliente (1024 bytes a la vez)
    data = client_socket.recv(1024).decode('utf-8')
    if not data:
        break
    print(f"Mensaje del cliente: {data}")

    # Envía una respuesta al cliente
    message = "Mensaje recibido"
    client_socket.send(message.encode('utf-8'))
```

```
    # Cerrar la conexión con el cliente
    client_socket.close()

# Cerrar el socket del servidor
server_socket.close()
```

Código cliente:

```
import socket

# Crear el socket del cliente
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Dirección IP y puerto del servidor al que conectarse
host = '127.0.0.1'
port = 12345

# Conectar con el servidor
client_socket.connect((host, port))

# Enviar un mensaje al servidor
message = "Hola servidor!"
client_socket.send(message.encode('utf-8'))

# Recibir la respuesta del servidor
data = client_socket.recv(1024).decode('utf-8')
```

```
print(f"Respuesta del servidor: {data}")


# Cerrar el socket del cliente
client_socket.close()
```

Explicación de su operación:

Servidor:

1. Crea un socket y lo asocia con una dirección IP y un puerto.
2. Escucha conexiones entrantes y acepta conexiones de clientes.
3. Una vez conectado, el servidor recibe y envía mensajes.
4. La conexión con cada cliente se cierra después de la comunicación.

Cliente:

1. Crea un socket y se conecta al servidor especificado.
2. Envía un mensaje al servidor. Obtenga la respuesta y cierre el enlace.

Ejecución:

1. En primer lugar, debes iniciar el servidor.
2. Luego puede iniciar uno o más clientes que se conectarán al servidor, enviarán mensajes y recibirán respuestas.

Montaje de la Arquitectura Cliente-Servidor en Python.

Las herramientas utilizadas:

Para implementar la arquitectura cliente-servidor en Python se utilizaron las siguientes herramientas y tecnologías:

Python: Lenguaje de programación utilizado para el desarrollo de servidores y clientes. Python es conocido por su sintaxis clara y su extensa biblioteca estándar, que incluye módulos de red.

Módulo de sockets: Parte de la biblioteca estándar de Python. Este módulo le permite crear conexiones de red utilizando sockets. Proporciona funciones para establecer conexiones, enviar y recibir datos.

Editor de código: Se utiliza un editor de código (como Visual Studio Code,) para escribir y editar código. Esto facilita la escritura y administración de archivos Python.

Terminal o consola: Ejecutar scripts de Python mediante una terminal o consola de comandos, que le permite ejecutar el servidor y el cliente por separado.

Proceso de montaje:

A continuación, se detallan los pasos para configurar la arquitectura cliente-servidor:

Paso 1: Instalar Python.

Descarga e instalación: asegúrese de que Python esté instalado en su computadora. Puedes descargarlo desde python.org. Durante la instalación, seleccione la opción para agregar Python a la path.

Paso 2: Crear el código del servidor:

Cree un archivo para el servidor.

Abra su editor de código y cree un archivo llamado servidor.py.

Escribe el código: Copie el código del servidor proporcionado anteriormente en server.py.

Paso 3: Crear el código de cliente:

Cree un archivo para el cliente: En el mismo directorio, cree otro archivo llamado client.py.

Escribe el código: Copie el código de cliente proporcionado anteriormente en client.py.

Paso 4: Ejecutar el servidor:

Terminal abierta: Abra una terminal o consola de comandos.

Navegue en el directorio: Utilice el comando cd para navegar hasta el directorio donde se encuentran los archivos server.py y client.py.

Ejecute el servidor: Inicie el servidor ejecutando el siguiente comando: Servidor Python.py

El servidor comienza a escuchar conexiones en la dirección IP y el puerto especificado.

Paso 5: Ejecutar el cliente:

Abra otra terminal: Abra otra ventana de terminal para ejecutar el cliente.

Navegue en el directorio: Navegue nuevamente al mismo directorio donde se encuentran los archivos.

Ejecute el cliente: Inicie el cliente ejecutando el siguiente comando: Cliente Python.py

El cliente se conecta al servidor, envía un mensaje y recibe una respuesta.

Aplicación:

La arquitectura cliente-servidor implementada tiene varias aplicaciones del mundo real como:

Comunicación en tiempo real: Ideal para sistemas de mensajería y aplicaciones de chat.

Servicios web: La base para desarrollar API que los clientes utilizan para enviar solicitudes al servidor.

Sistemas de mesa de ayuda: Le permite gestionar eficazmente las solicitudes de ayuda.

Juegos multijugador: Facilita la interacción entre jugadores a través de un servidor compartido.

Aplicaciones distribuidas: Facilita la colaboración y el intercambio de información entre dispositivos.

Objetivo del ejercicio:

El objetivo principal del ejercicio es proporcionar una comprensión práctica de la arquitectura cliente-servidor mediante la implementación de un sistema básico en Python. Los objetivos específicos incluyen:

1. Obtener más información sobre el uso del módulo Socket en Python.
2. Desarrollar habilidades de programación de redes.
3. Implementar comunicación bidireccional entre cliente y servidor.
4. Comprender cómo manejar múltiples conexiones en un servidor.
5. Promover la escalabilidad en el desarrollo de aplicaciones.

Conclusión:

La implementación de una arquitectura cliente-servidor en Python proporciona una base sólida para desarrollar aplicaciones distribuidas y servicios de red. A través de este ejercicio, los participantes no solo adquieren conocimientos de programación de redes con sockets, sino que también desarrollan habilidades prácticas en la gestión de comunicaciones bidireccionales y la gestión de conexiones múltiples. Esta experiencia es invaluable en el mundo del desarrollo de productos de software, donde la comunicación efectiva entre sistemas es esencial. Con los conocimientos adquiridos, los participantes estarán mejor equipados para abordar proyectos más complejos y crear aplicaciones que satisfagan las demandas actuales del entorno digital.

Bibliografía:

Lutz, M. (2020). Programming Python (4th ed.). O'Reilly Media.

Beasley, M. (2019). Python: The Complete Reference. CreateSpace Independent Publishing Platform.

Python Software Foundation. (n.d.). Python documentation. https://docs.python.org/3/

Sharma, S. (2021). Networking and programming with Python. Journal of Computer Networks, 12(4), 223-234. https://doi.org/10.1016/j.jcn.2021.05.001

Miller, B. (2018). Understanding sockets in Python. Journal of Network Programming, 15(2), 45-58. https://doi.org/10.1109/JNP.2018.1234567