Semana N° 02

Guía Práctica de Laboratorio N°2

Taller práctico sobre sockets y modelos de arquitecturas de sistemas distribuidos

---

Sección:..……..…Apellidos y nombres:.....
Docente:..... Fecha: ..... / ..... / 202…

Duración:.....min. Tipo de práctica: Individual ( ) Equipo ( )

---

1. **Instrucciones**

   El estudiante desarrollará un sistema Cliente-Servidor en Python para la consulta de inventario. El servidor alojará una base de datos SQLite y responderá a consultas de productos realizadas desde otro equipo cliente en una red LAN mediante sockets.

2. **Propósito / Objetivo:**

   Desarrollar habilidades en la implementación de aplicaciones distribuidas que:

   - Usan SQLite como base de datos embebida.
   - Emplean sockets para la comunicación en red.
   - Funcionan correctamente en una red local entre dos máquinas.

3. **Fundamento Teórico:**

   a. Arquitectura Cliente-Servidor:

   Es un modelo donde un cliente solicita servicios o recursos a un servidor. El servidor procesa la solicitud y devuelve una respuesta.

   b. Sockets en Python:

   Un socket es un punto de comunicación que permite enviar y recibir datos entre dispositivos en red. Python proporciona el módulo socket para este propósito.

   c. SQLite:

   Es una base de datos ligera y embebida que no requiere instalación ni configuración. Ideal para prototipos o aplicaciones pequeñas.

4. **Equipos, Software, Materiales y Reactivos** (según sea el caso)

   Equipos:

   - 2 computadoras conectadas a la misma red (una como servidor, otra como cliente)

   Software:

   - Python 3.x
   - SQLite (viene incluido con Python)
   - Editor de código Visual Studio con complemento de python instalado

   Material:

   - Red LAN estable (WiFi o cableada)

5. Indicaciones / Instrucciones previas:

- Instalar Python 3 en ambos equipos.
- Verificar que estén en la misma red (verifica con ping).
- Habilitar el puerto 5000 en el firewall del servidor.
- Verifica la IP local del servidor (con ipconfig en Windows o ifconfig en Linux/macOS).

6. Procedimientos:

Primero: Crear base de datos SQLite (setup_db.py)

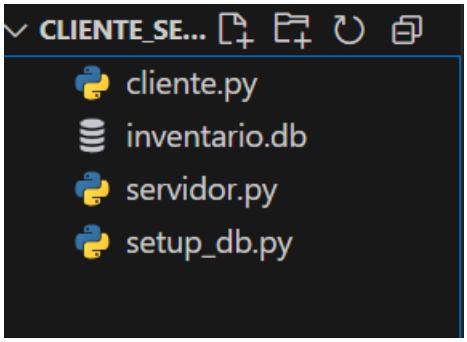
```
import sqlite3  # Módulo para trabajar con bases de datos SQLite

# Crear (o conectar si ya existe) la base de datos local

conn = sqlite3.connect('inventario.db')

cursor = conn.cursor()

# Crear la tabla de productos si no existe

cursor.execute('''

CREATE TABLE IF NOT EXISTS productos (

    id INTEGER PRIMARY KEY,

    nombre TEXT NOT NULL,

    cantidad INTEGER NOT NULL

)

''')

# Insertar productos de ejemplo

cursor.executemany("INSERT INTO productos (nombre, cantidad)
VALUES (?, ?)", [

    ("Teclado", 10),

    ("Mouse", 25),

    ("Monitor", 5)

])

# Guardar cambios y cerrar conexión

conn.commit()
```

```
conn.close()
```

Ejecuta el servidor con el comando

```
Python setup_db.py
```

Observa como se crea la base de datos inventario



Segundo: Código del Servidor (servidor.py)

```
import socket        # Para comunicación en red
import sqlite3       # Para conectarse a SQLite
import threading        # Para manejar múltiples clientes simultáneamente


HOST = '0.0.0.0'    # Acepta conexiones desde cualquier interfaz de red
PORT = 5000          # Puerto donde se aceptan conexiones TCP


# Función que atiende a cada cliente de forma independiente
def atender_cliente(conn, addr):
    print(f"[+] Conectado con {addr}")


    try:
        # Recibir datos del cliente (máximo 1024 bytes)
```

```
        data = conn.recv(1024).decode()
        print(f"[{addr}] Consulta recibida: {data}")

        # Conectar con base de datos y buscar producto
        conexion = sqlite3.connect('inventario.db')
        cursor = conexion.cursor()
        cursor.execute("SELECT cantidad FROM productos WHERE
nombre = ?", (data,))
        resultado = cursor.fetchone()
        conexion.close()

        # Preparar respuesta
        if resultado:
            respuesta = f"Cantidad disponible: {resultado[0]}"
        else:
            respuesta = "Producto no encontrado"

        # Enviar respuesta
        conn.send(respuesta.encode())

    except Exception as e:
        print(f"[{addr}] Error: {e}")
        conn.send("Error en la consulta".encode())
    finally:
        conn.close()
        print(f"[-] Conexión con {addr} cerrada")
```

```
# Crear socket del servidor
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
servidor.bind((HOST, PORT))
servidor.listen(5)  # Acepta hasta 5 conexiones pendientes

print(f"Servidor en ejecución. Escuchando en {HOST}:{PORT}...")

# Bucle principal para aceptar clientes
while True:
    conn, addr = servidor.accept()  # Esperar conexión
    # Crear un hilo para manejar al cliente sin bloquear el servidor
    hilo = threading.Thread(target=atender_cliente, args=(conn, addr))
    hilo.start()
```

Ejecuta el cliente con el comando:

```
Python servidor.py
```

Tercero: Código del Cliente (cliente.py)

```
import socket  # Módulo para sockets en Python

# IP del servidor (cambiar a la IP real del equipo servidor en la LAN)
HOST = '192.168.1.50'
```

```
PORT = 5000  # Debe coincidir con el puerto usado por el servidor

# Crear socket TCP (SOCK_STREAM) con protocolo IPv4 (AF_INET)
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Conectar al servidor usando su IP y puerto
cliente.connect((HOST, PORT))

# Solicitar al usuario el nombre del producto a consultar
producto = input("Ingrese el nombre del producto: ")

# Enviar el producto codificado en bytes
cliente.send(producto.encode())

# Recibir respuesta del servidor (máximo 1024 bytes)
# 1024 es el tamaño del buffer, suficiente para mensajes pequeños
respuesta = cliente.recv(1024).decode()

# Mostrar la respuesta del servidor
print(f"Respuesta del servidor: {respuesta}")

# Cerrar conexión con el servidor
cliente.close()
```

Ejecuta el cliente con el comando:

```
Python cliente.py
```

Se pedirá un nombre del producto, proporcionamos el nombre y damos Enter
¿Cuál fue la respuesta del servidor?

7. Resultados

Durante el desarrollo de la práctica, se espera lograr los siguientes resultados:

- Configuración y ejecución exitosa de un sistema Cliente-Servidor utilizando sockets TCP sobre una red LAN.
- Creación de una base de datos local SQLite, cargada con productos de prueba, que es consultada en tiempo real por clientes remotos.
- Interacción entre dos equipos físicos, donde uno actúa como servidor de datos y otro como cliente de consultas, validando la implementación de una arquitectura distribuida básica.

Estos resultados evidencian que los estudiantes han comprendido cómo diseñar y desplegar una arquitectura Cliente-Servidor funcional en red local, apoyada por una base de datos integrada.

8. Conclusiones

- Se reforzó el uso del modelo Cliente-Servidor en la vida real, simulando un escenario práctico donde un servidor provee información a múltiples clientes.
- El uso de SQLite mostró cómo una base de datos local puede integrarse fácilmente en soluciones pequeñas o prototipos sin sacrificar funcionalidad.
- Se comprobó que una simple arquitectura distribuida es suficiente para realizar tareas útiles como la consulta remota de inventario, permitiendo escalar o mejorar el sistema en versiones futuras.

9. Sugerencias y/o recomendaciones

- Usa try/except para manejar errores como pérdida de conexión.
- Si usas IP dinámica, considera configurar IP fija o nombre DNS local.