Semana N° 02

Guía Práctica de Laboratorio N°3

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**

   En esta práctica, desarrollarás un sistema P2P (Peer-to-Peer) básico que permitirá a tres equipos conectados en la misma red LAN compartir archivos entre sí sin depender de un servidor central. Cada equipo podrá:

   - Enviar archivos a otros nodos.
   - Recibir archivos de otros nodos.
   - Registrar y consultar archivos disponibles.

2. **Propósito / Objetivo:**

   Desarrollar habilidades en la implementación de aplicaciones distribuidas que:

   - Comprender el modelo P2P en sistemas distribuidos.
   - Desarrollar un sistema descentralizado para compartir recursos.
   - Aplicar sockets y multithreading en redes LAN.
   - Implementar una lógica de descubrimiento y transferencia de archivos.

3. **Fundamento Teórico:**

   Un sistema P2P (peer-to-peer) distribuye la carga y responsabilidad entre los nodos. A diferencia del modelo cliente-servidor, cada nodo puede actuar como cliente y servidor. En esta práctica:

   - Cada nodo corre un proceso que escucha peticiones y envía archivos.
   - Se utiliza TCP para garantizar la transferencia completa de archivos.
   - Se pueden compartir archivos comunes como imágenes, textos o documentos PDF.

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

   Equipos:

   - 3 computadoras conectadas a la misma red LAN.

   Software:

   - Python 3.x
   - Editor de código Visual Studio con complemento de python instalado

   Material:

   - Red LAN estable (WiFi o cableada)
   - Archivos de prueba (PDF, .txt, .jpg, etc.).

5. Indicaciones / Instrucciones previas:

- Asegúrate de que todos los equipos estén en la misma red.
- Establece IPs fijas o toma nota de sus IPs locales.
- Comparte una carpeta con algunos archivos por cada nodo para la prueba.
- Revisa que los puertos usados estén habilitados por el firewall.

6. Procedimientos:

Primero: Crear Estructura del nodo P2P (nodo_p2p.py)

```
import socket

import threading

import os



# CONFIGURACIÓN DEL NODO

IP_LOCAL = '0.0.0.0'          # Escuchar en todas las interfaces

de red

PUERTO = 6000                   # Puerto que usa el sistema P2P

CARPETA_ARCHIVOS = 'compartidos'  # Carpeta compartida



# Crear carpeta si no existe

if not os.path.exists(CARPETA_ARCHIVOS):

    os.makedirs(CARPETA_ARCHIVOS)



# Lista de peers (IPs de otros nodos en la red)

peers = [

    ('192.168.1.10', 6000),

    ('192.168.1.11', 6000),

    ('192.168.1.12', 6000)

]
```

```
# ------------------------ SERVIDOR DEL NODO ----------------
------------


# Escucha y atiende múltiples conexiones
def atender_conexiones():
    servidor          =          socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    servidor.bind((IP_LOCAL, PUERTO))
    servidor.listen(5)
    print(f"[ESCUCHANDO] Nodo activo en puerto {PUERTO}")

    while True:
        conn, addr = servidor.accept()
        print(f"[CONEXIÓN] Desde {addr}")
        threading.Thread(target=gestionar_peticion,
args=(conn,)).start()

# Atiende cada tipo de solicitud recibida
def gestionar_peticion(conn):
    try:
        solicitud = conn.recv(1024).decode()

        # Buscar archivo
        if solicitud.startswith("BUSCAR:"):
            nombre = solicitud.split(":", 1)[1]
            ruta = os.path.join(CARPETA_ARCHIVOS, nombre)
```

```
            if os.path.exists(ruta):
                conn.send(b"SI")
            else:
                conn.send(b"NO")

        # Descargar archivo
        elif solicitud.startswith("DESCARGAR:"):
            nombre_archivo = solicitud.split(":", 1)[1]
            ruta = os.path.join(CARPETA_ARCHIVOS,
nombre_archivo)
            if os.path.exists(ruta):
                conn.send(b"OK")
                with open(ruta, 'rb') as f:
                    while True:
                        datos = f.read(1024)
                        if not datos:
                            break
                        conn.sendall(datos)
                print(f"[ENVÍO COMPLETO] {nombre_archivo}")
            else:
                conn.send(b"NOFILE")

    except Exception as e:
        print(f"[ERROR] {e}")
    finally:
        conn.close()
```

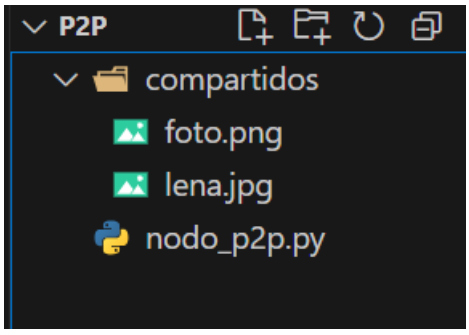
```
# ------------------------ CLIENTE DEL NODO -----------------
-----------


# Busca en todos los peers si tienen el archivo
def buscar_en_peers(nombre_archivo):
    disponibles = []
    for ip, puerto in peers:
        try:
            s = socket.socket()
            s.settimeout(2)
            s.connect((ip, puerto))
            s.send(f"BUSCAR:{nombre_archivo}".encode())
            respuesta = s.recv(1024)
            if respuesta == b"SI":
                disponibles.append((ip, puerto))
            s.close()
        except:
            pass
    return disponibles


# Descarga el archivo desde el peer seleccionado
def descargar_archivo(ip_destino, nombre_archivo):
    try:
        cliente = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

```
        cliente.connect((ip_destino, PUERTO))
        cliente.send(f"DESCARGAR:{nombre_archivo}".encode())

        respuesta = cliente.recv(1024)
        if respuesta == b"OK":
            ruta_destino = os.path.join(CARPETA_ARCHIVOS,
nombre_archivo)
            with open(ruta_destino, 'wb') as f:
                while True:
                    datos = cliente.recv(1024)
                    if not datos:
                        break
                    f.write(datos)
            print(f"[DESCARGADO EN COMPARTIDOS]
{nombre_archivo}")
        else:
            print("[NO ENCONTRADO] El archivo no está
disponible.")
        cliente.close()

    except Exception as e:
        print(f"[ERROR AL DESCARGAR] {e}")

# ---------------------- INICIO DEL SISTEMA -----------------
-----------
```

```
# Lanzar servidor en un hilo aparte
threading.Thread(target=atender_conexiones,
daemon=True).start()


# Menú principal
while True:
    print("\n--- MENÚ P2P ---")
    print("1. Ver archivos compartidos")
    print("2. Buscar y descargar archivo")
    print("3. Salir")
    opcion = input("Seleccione una opción: ")


    if opcion == '1':
        print("\nArchivos disponibles en este nodo:")
        for f in os.listdir(CARPETA_ARCHIVOS):
            print(" -", f)


    elif opcion == '2':
        archivo = input("Nombre del archivo que deseas buscar:
")
        disponibles = buscar_en_peers(archivo)


        if not disponibles:
            print("No se encontró ese archivo en la red.")
        else:
            print("Archivo disponible en:")
```

```
            for idx, (ip, _) in enumerate(disponibles):
                print(f"{idx+1}. {ip}")
            eleccion = int(input("Selecciona de dónde deseas
descargar (número): "))
            ip_seleccionada = disponibles[eleccion - 1][0]
            descargar_archivo(ip_seleccionada, archivo)

    elif opcion == '3':
        print("Saliendo...")
        break
    else:
        print("Opción inválida.")
```

Crea la carpeta compartida donde compartirás y recibirás archivos, para comenzar puedes poner dos imágenes.



En cada cliente/server ejecuta el nodo

```
Python nodo_p2p
```

7. Resultados

- Cada nodo puede ejecutar el programa y quedar escuchando peticiones de otros.
- Se puede solicitar archivos a cualquiera de los otros nodos.
- Los archivos se descargan correctamente en la máquina solicitante.
- El sistema funciona sin un servidor central y los nodos se comunican directamente.

8. Conclusiones

- Se implementó con éxito un sistema distribuido P2P básico para compartir archivos entre múltiples nodos.
- Cada nodo fue capaz de actuar como cliente y servidor simultáneamente.
- Se validó la utilidad del modelo P2P para distribuir cargas de trabajo y descentralizar el control.
- El uso de threading permitió que cada nodo atendiera múltiples conexiones sin bloquear su funcionalidad.
- Se desarrolló un sistema robusto y extensible que puede adaptarse a más nodos y funcionalidades avanzadas como autenticación o búsqueda automática.

9. Sugerencias y/o recomendaciones

- Utilizar nombres de archivo únicos para evitar sobreescritura.
- Implementar autenticación básica o listas de control de acceso para mayor seguridad.
- Usar una estructura de directorios organizada para los archivos recibidos.
- Para producción, considerar cifrado y validación de integridad.