

Actividad Evaluativa - Eje 3 - Hilos y Socket

Melqui A. Romero y Karolaine Z. Villero

Universidad Área Andina, Bogotá D.C

Fundación Universitaria del Área Andina

Facultad De Ingeniería y Ciencias Básicas, Ingeniería en Sistemas,

Modelos de Programación II

Ing. Deivys Morales

16 de marzo del 2025

Actividad Evaluativa - Eje 3 - Hilos y Socket

La implementación de sistemas de comunicación en red utilizando Hilos y Sockets en Python es fundamental para el desarrollo de aplicaciones Cliente-Servidor modernas. Los Sockets permiten establecer conexiones fiables entre dispositivos, mientras que los Hilos permiten gestionar múltiples clientes de manera simultánea sin bloquear la ejecución principal del programa. Este trabajo presenta un análisis detallado de cómo se implementa un servidor TCP multicliente en Python y su interacción con clientes que realizan operaciones con estructuras de datos tipo Árbol Binario. La comunicación se establece mediante el protocolo TCP, utilizando la librería estándar socket, y la concurrencia se maneja con la librería threading. Además, se destaca la importancia de la sincronización para garantizar la integridad de los datos y la eficiencia en la comunicación.

Tabla de contenido

Tablas de Figuras	4
1. Objetivo General	5
2. Objetivos Específicos.....	5
3. Hilos y socket.....	6
Hilos.	6
Socket	6
2. Estructura general del código.....	8
2.1 Descripción	8
Ejercicio 1:.....	8
Ejercicio 2:.....	8
Aplicación principal	9
Ejecución del programa	9
3.1. Como ejecutarlo	9
3.2 Salidas	10
3.3. Video explicativo.....	10
Conclusiones	11
Referencias	12

Tablas de Figuras

Figura 1 Código tipo Hilo	6
Figura 2: Código tipo Socket	7
Figura 3 Estructura del proyecto	9
Figura 4 Menú inicial.....	9
Figura 5 Datos insertados – Opción 4.....	10

1. Objetivo General

Desarrollar e implementar un sistema de comunicación Cliente-Servidor utilizando Hilos y Sockets en Python, que permita gestionar múltiples conexiones simultáneamente y realizar tareas específicas con cada cliente, incluyendo el manejo de un Árbol Binario.

2. Objetivos Específicos

1. Implementar un servidor TCP multicliente que permita conexiones concurrentes usando hilos.
2. Crear un cliente TCP que se conecte al servidor y realice operaciones específicas.
3. Utilizar un Árbol Binario para almacenar y organizar datos.
4. Evaluar la eficiencia de la comunicación Cliente-Servidor y el manejo de hilos.

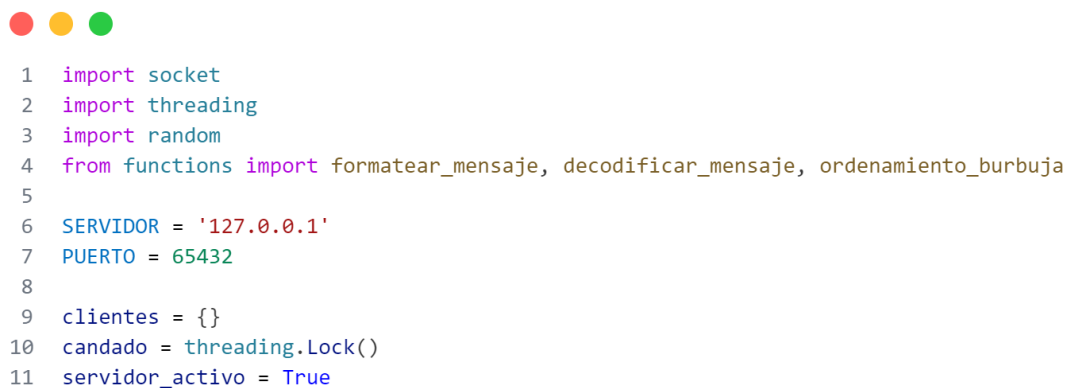
3. Hilos y socket

Hilos.

La programación de hilos permite la ejecución simultánea de tareas dentro de un programa. Un hilo es una unidad de control que opera con su propio contador y pila de ejecución. A diferencia de los procesos, que son independientes y se comunican mediante mecanismos específicos, los hilos comparten recursos directamente, lo que facilita la concurrencia en los sistemas operativos: (Tavares)

Figura 1 Código tipo Hilo

Esta imagen contiene programación Socket, el cual es parte de código del programa



```
1 import socket
2 import threading
3 import random
4 from functions import formatear_mensaje, decodificar_mensaje, ordenamiento_burbuja
5
6 SERVIDOR = '127.0.0.1'
7 PUERTO = 65432
8
9 clientes = {}
10 candado = threading.Lock()
11 servidor_activo = True
```

Socket

La programación de sockets en Java permite la comunicación Cliente-Servidor, tratando las redes como entrada y salida de archivos. Un socket es un punto final de conexión en la red, permitiendo el intercambio de datos. (Oracle, s.f.)

Existen dos tipos principales:

- **Sockets de flujo (TCP):** Conexión fiable, garantiza el orden y la entrega de datos (ej. transferencia de archivos). (IBM, s.f.)
- **Sockets de datagrama (UDP):** Envía paquetes sin garantía de entrega, ideal para transmisiones rápidas (ej. video en vivo). (Onmex, s.f.)

Java usa las clases Socket y ServerSocket para la implementación de conexiones en red.

Figura 2: Código tipo Socket

Esta imagen contiene programación Socket, el cual es parte de código del programa



```

1  def jugar(cliente, numero, disponibles):
2      print(f"\nNúmero seleccionado: {numero}")
3
4      cliente.send(formatear_mensaje(','.join(map(str, disponibles))))
5
6      for intento in range(3):
7          try:
8              respuesta = decodificar_mensaje(cliente.recv(1024))
9              print(f"Respuesta del servidor: {respuesta}")
10
11              if "¡El número es" in respuesta:
12                  return False
13
14              if intento < 2:
15                  input("\nPresione Enter para el siguiente intento...")
16                  cliente.send(formatear_mensaje("siguiente"))
17          except socket.error as e:
18              print(f"Error de conexión: {e}")
19              return True
20
21      try:
22          resumen = decodificar_mensaje(cliente.recv(1024))
23          print(resumen)
24      except socket.error as e:
25          print(f"Error al recibir resumen: {e}")
26          return True
27
28      return "Perdiste" in resumen

```

2. Estructura general del código

2.1 Descripción

El proyecto se divide en tres ejercicios principales, más una aplicación principal, la cual mostramos a continuación.

Ejercicio 1:

Clases: functions

Funciones principales:

- Genera números aleatorios (1 a 10) y los agrega a una lista si tiene menos de 5 elementos.
- Usa hilos para agregar números a la lista cada segundo.
- Intenta ordenar la lista con Burbuja (pero tiene un error en la línea de intercambio).
- Codifica y decodifica mensajes a formato UTF-8.

Clase: functions (menú interactivo para las operaciones anteriores)

Ejercicio 2:

Clases: Server

Funciones principales:

- Adivina números enviados por clientes en 3 intentos.
- Usa hilos para manejar varias conexiones al mismo tiempo.
- Permite apagar el servidor con el comando apagar.

Clase: Server (menú interactivo para las operaciones anteriores)

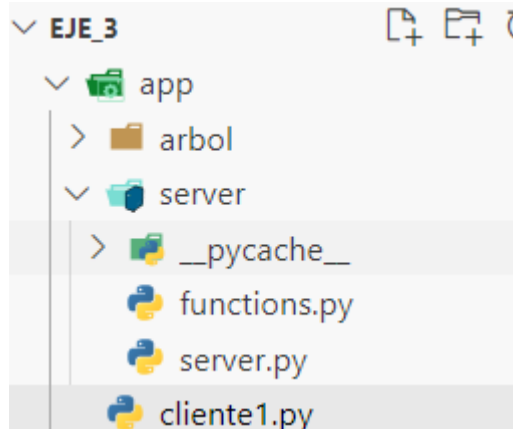
Aplicación principal

Permite al usuario realizar consultas:

- Se conecta al servidor (127.0.0.1:65432) para jugar a adivinar números.
- Usa un Árbol Binario para guardar y ordenar números.
- Permite agregar números, ver el árbol, jugar o salir.
- Envía y recibe mensajes en UTF-8.

Figura 3 Estructura del proyecto

Estructura de creación y desarrollo de Eje 3



Nota: Esta imagen es el capture de la estructura del Eje Hilo y stock.

Ejecución del programa

3.1. Como ejecutarlo

Para la ejecución del programa debemos ejecutar el código en el archivo *cliente1.py* este una vez ejecutado iniciará la App.

Figura 4 Menú inicial

Menú ejecutado del programa Python

```
PS C:\xampp\htdocs\universidadAreandina-main\Semestre5\modelos de programación\ej_3> & C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/xampp\htdocs/universidadAreandina-main/Semestre5/modelos de programación/eje_3/app/cliente1.py"
Intentando conectar al servidor 127.0.0.1:65432...
¡Conectado al servidor desde el puerto local 54452!

** MENÚ DEL ÁRBOL **
1. Agregar número
2. Ver árbol (orden ascendente)
3. Ver estructura del árbol
4. Jugar a adivinar número
5. Salir
Seleccione una opción:
```

Nota: Menú desplegado después de ejecutar el código.

3.2 Salidas

Las salidas que puede generarse al ejecutar el flujo son muy diversas ya que la dinámica es algo amplia, a continuación, se mostrara un resultado.

Figura 5 Datos insertados – Opción 4

Resultado de la ejecución del código.

```
** MENÚ DEL ÁRBOL **
1. Agregar número
2. Ver árbol (orden ascendente)
3. Ver estructura del árbol
4. Jugar a adivinar número
5. Salir
Seleccione una opción: 4

Números disponibles: [7, 12, 34, 45, 65, 90]

Seleccione un número: 90

Número seleccionado: 90
Respuesta del servidor: ¡El número es 90! Lo adiviné en el intento 1 de 3
```

Nota: Resultado después de ejecutar con la opción 4 el comando realizado.

3.3. Video explicativo

Para la comprensión del programa se ha realizado un video explicativo, donde se muestra desde la estructura del programa hasta la ejecución del mismo. El cual fue subido a la plataforma

YouTube: <https://youtu.be/2ZWtwF7MPGs>

Conclusiones

El presente trabajo demuestra la efectividad del uso de Hilos y Sockets en la construcción de aplicaciones Cliente-Servidor con Python. A través de la implementación de un servidor TCP multicliente, se evidenció cómo los hilos permiten manejar múltiples conexiones simultáneamente sin bloquear la ejecución principal del programa, lo cual es esencial para sistemas que requieren escalabilidad y eficiencia.

El uso de sockets TCP proporcionó un mecanismo confiable para la transmisión de datos entre clientes y servidores, permitiendo una comunicación segura y ordenada. Asimismo, se logró implementar un Árbol Binario en el cliente para gestionar datos de manera estructurada y eficiente, aprovechando sus ventajas en la organización y recuperación de información.

Además, se resaltó la importancia de la sincronización mediante `threading.Lock()` para proteger recursos compartidos y evitar conflictos durante la concurrencia. Este trabajo sirve como base para desarrollar aplicaciones de red más complejas, donde la combinación de Hilos y Sockets es esencial para garantizar la integridad de los datos y la correcta interacción entre múltiples clientes.

En conclusión, la implementación desarrollada demuestra cómo estas tecnologías pueden aplicarse efectivamente en diversos escenarios que requieren comunicación en red y procesamiento concurrente.

Referencias

- IBM. (s.f.). *Tipos de socket*. Obtenido de <https://www.ibm.com/docs/es/aix/7.3?topic=protocols-socket-types>.
- Meneses, E. (26 de 11 de 2021). *Clase8-Arboles.pdf*. Obtenido de ESTRUCTURAS DE DATOS: <https://www.uv.mx/personal/ermeneses/files/2021/08/Clase8-Arboles.pdf>
- Onmex. (s.f.). *¿Qué es un socket?* Obtenido de <https://onmex.mx/tecnologia-y-desarrollo/que-es-un-socket/>.
- Oracle. (s.f.). *All About Sockets (The Java™ Tutorials > Custom Networking)*. Obtenido de <https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>.
- Silvia Guardati, O. (1 enero 2006). *Estructuras de Datos - 3ra Edición*. McGraw Hill Education.
- Tavares, G. E. (s.f.). <https://areandina.instructure.com/courses/57252>. Obtenido de Modelos de Programación II – Eje 3: Pongamos en práctica.