**WEB SERVER USING SOCKET PROGRAMMING**



**Prepared by:**

**Balbino Pedro Baptista 103012350551**

**Muhammad Rafii Munfaadi**

**Muhammad Zayyaad Azizan**

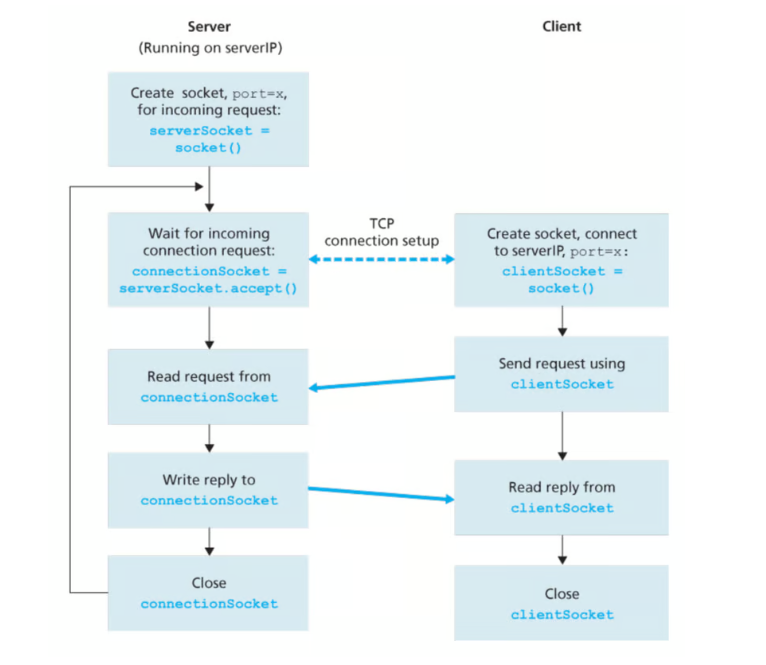
**Telkom University**

**205/2026**

## Socket Programming

This project aims to build a simple TCP-based web server that processes HTTP requests using both single-threaded and multi-threaded approaches. The server listens for incoming connections and, upon receiving an HTTP GET request, reads the requested HTML file from the local system and sends it back to the client over a TCP connection. If the requested file is not available, the server responds with a properly formatted HTTP 404 Not Found message, demonstrating basic web server functionality and the handling of static content delivery

The flow of how socket works :



1. **SPESIFICATIONS**

* **Basic**

The server is designed to accept HTTP GET requests from clients and handle them by first parsing the request to extract the filename being requested. Once the filename is identified, the server attempts to retrieve the corresponding file from the local file system. If the file is found, it is read and sent back to the client along with appropriate HTTP response headers, such as status code, content type, and content length, ensuring the client can correctly interpret the response. In cases where the requested file does not exist, the server gracefully handles the error by returning a 404 Not Found response, informing the client that the resource could not be located. This behavior demonstrates a foundational implementation of how HTTP servers interact with clients and serve static content

* **Single Thread**

The single-threaded version of the server handles only one client request at a time, meaning it must complete processing one client's request before moving on to the next. This approach, while limited in scalability, is valuable for educational purposes as it clearly demonstrates the fundamental concept of how a web server operates. The main function responsible for managing each client request is handle\_client\_connection, which processes the HTTP request, retrieves the appropriate file, and sends the response back to the client. This design makes it easier to understand the core mechanisms of request handling without the added complexity of concurrent execution.

* **Multi-Thread**

The multi-threaded version of the server is capable of serving multiple client requests simultaneously, greatly improving efficiency and responsiveness compared to the single-threaded approach. It achieves this by processing each incoming client connection in a separate thread using Python's threading module, allowing multiple clients to be handled in parallel without blocking one another. The main functionality is divided between two key functions: start\_server, which sets up the server socket and listens for incoming connections, and handle\_client, which manages the request and response cycle for each individual client. This design reflects a more realistic and scalable architecture used in real-world web servers.

1. **SOURCE CODE**

The link of the code: <https://github.com/balbinopb/final-project-jarkom>

* **Single Thread**

This Python script is a simple HTTP server that listens for connections, accepts HTTP GET requests, and sends back the requested file if it exists.

import socket: This line imports the socket module, which provides low-level networking interface.

socket: used to create and manage network connections.

os: used here to check if requested files exist (os.path.isfile()).



Defines the server's **IP address** and **port number**.  
127.0.0.1 means it runs locally only.

A close up of a text

AI-generated content may be incorrect.

socket.AF\_INET: IPv4 addressing.

socket.SOCK\_STREAM: TCP connection.

bind(('', server\_port)): Binds to all interfaces on the given port.

listen(1): Listens for incoming connections (1 queued connection).

A computer code with colorful text

AI-generated content may be incorrect.

Accepts a new client connection and Returns a **new socket** for communicating with the client and the **client's address**



Receive and Parse HTTP Request

A screen shot of a computer screen

AI-generated content may be incorrect.

Check and Serve File

A screenshot of a computer code

AI-generated content may be incorrect.

Send HTTP 200 OK Response

A screen shot of a computer program

AI-generated content may be incorrect.

If File Not Found – Send 404 Response

A screen shot of a computer code

AI-generated content may be incorrect.

* **Multi-Thread**

In the Multi-Thread the overall code is similar, but what makes different is, it’s using threading package package in order to handling for multiple request. Here is the code :



Enables creation of **multiple threads**, allowing the server to handle **multiple clients concurrently.**



Allows up to **5 queued incoming connections**, instead of just 1 in the single-threaded version

A computer screen with text

AI-generated content may be incorrect.

Creates and starts a **new thread** to handle each incoming client, so the server **doesn't block** while serving one request.



Now receives both the **connection object** and the **client's address**, to be compatible with multithreading and logging



Default to index.html if no file is requested



Reads HTML as **text** with UTF-8 encoding, compared to binary mode ('rb') in the single-threaded version



Sends a **500 Internal Server Error** if something unexpected happens.

* **Client**

import socket, sys: This line imports the socket and sys modules. The socket module is for network communications, and the sys module is for accessing command-line arguments.

A computer code with colorful text

AI-generated content may be incorrect.

Gets the **server IP**, **port**, and **file name** to request from the command line

A screenshot of a computer program

AI-generated content may be incorrect.

For this block connects the client to the server using the provided host and port, sends an HTTP GET request for the specified file, receives the response in chunks, decodes it, checks if the response status is "200 OK" (indicating success), and prints the URL if successful or an error message otherwise; it also handles any exceptions that occur during the process and ensures the socket is closed afterward.

1. **RESULTS**

The Results will be seen in the steps below:

* single thread

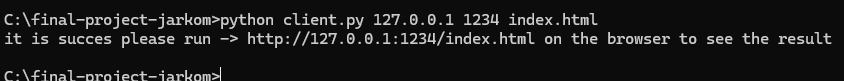
When the server is running :

A black screen with white text

AI-generated content may be incorrect.

This is single thread

After that we run client:



It give us URL then we open it on browser:

A computer screen shot of a blue background

AI-generated content may be incorrect.

The result

* Mutli thread

Multi thread server is running:

A black screen with white text

AI-generated content may be incorrect.

Run client:

A screenshot of a computer program

AI-generated content may be incorrect.

The result:

Index2.html

A blue screen with white text

AI-generated content may be incorrect.

Index3.html

A blue screen with white text

AI-generated content may be incorrect.

1. **CONCLUSIONS**

This project successfully implements two versions of a TCP web server handling simple HTTP functionality: a single-threaded version for basic understanding and a more efficient multi-threaded version capable of handling multiple clients concurrently. It demonstrates how socket programming combined with multithreading is essential for building real-world networked applications.