DCN FILE TRANSFER SYSTEM USING PYTHON SOCKET PROGRAMMING

Doucemntation

GROUP NO: 5

MUHAMMAD DAWOOD

2023-AG-9617

AYAAN KHURRAM

2023-AG-9611

ADINA ABRAR

2023-AG-9487

Submitted To: Dr.Ahsan Raza Sattar

Course Code: CS-406

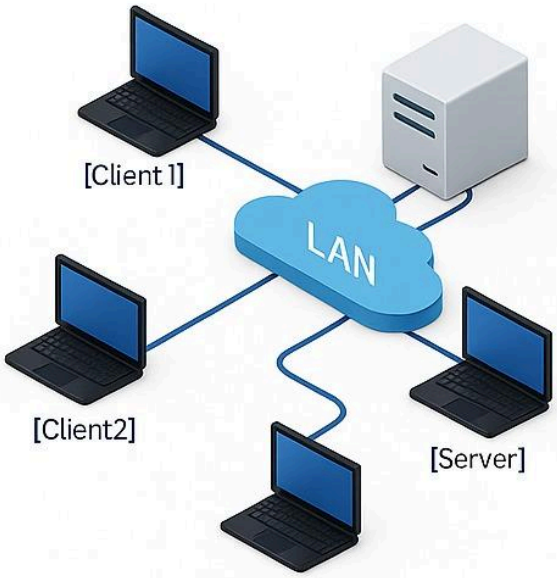
Course Title: DCN

DCN FILE TRANSFER SYSTEM USING PYTHON SOCKET PROGRAMMING

1 INTRODUCTION

- Uploading and sharing files over a network is essential for applications like remote backups and distributed file sharing.
- This project demonstrates a file transfer systm using Python's socket prograinming to communicate between clients and a server.

2 SYSTEM ARCHITECTURE



- One server, multiple clients setup is used for the file transfer system.
- Clients initiats the conection to server to upload or receive files.
- All devices communicate over a local area network (LAN).

OBJECTIVES

1 Efficient Data Sharing

- Quick and reliable quckly and reliable large files through du.

2 Secure File Transfer

- Encryption and hash-checking mechanisms for safegue-rding sensitivity.

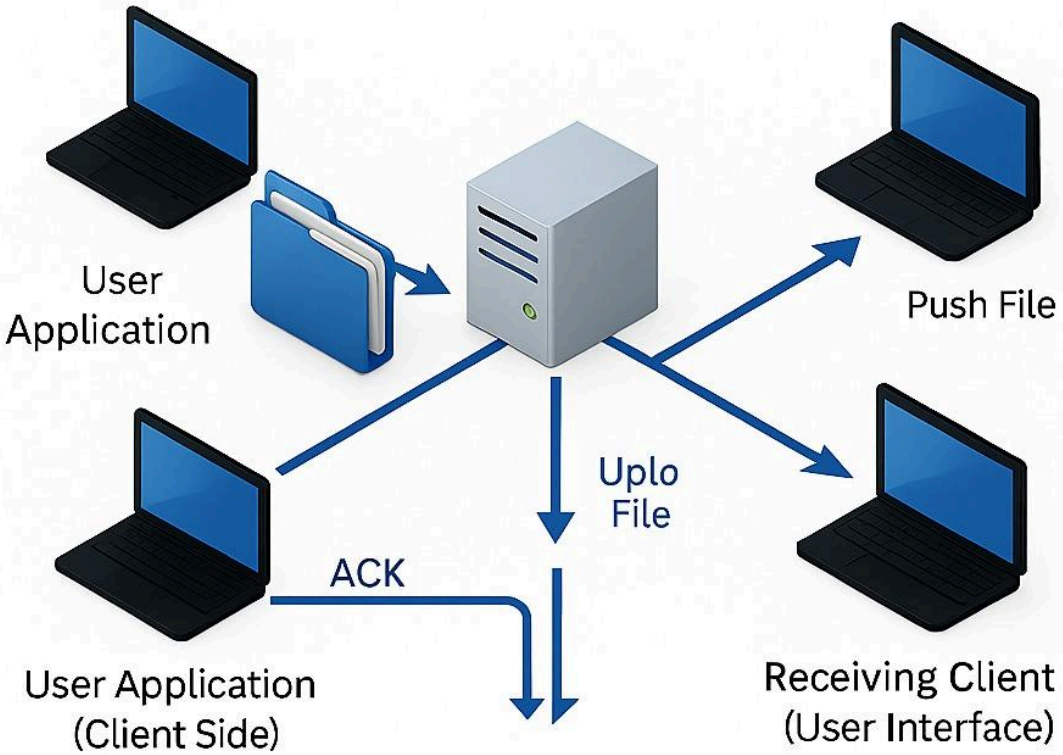
3 Scalability for Multiple Clients

- Multithreading on sgrvver to handle multiple uploads simultaneo.

4 Reliable Delivery and Acknowledgment

- Use of knnnhoxnesseges to confirm receipt receipt.

File Transfer Flow



PYTHON SOCKET FROGRAMMING

1. FILE TRANSSFER SYSTEM

Socket programming in Python allows computers to communicate over a network. It's widely used to implement client-server applications — and one of the most practical examples is a File Transfer System.

HOW IT WORKS:

- **Socket Creation**
  Python provides thesoket module to create a socket
- **Server Side**
  – Binds to a specific IP address and port
  – Listens for inconnections from clients
  – Accepts a connetion and waits to receive files
- **Client Side**
  – Connects to server's IP address and port

DATA FLOW EXAMPLE:

- Client. Selects a file and sends it in binary chunks
- **Server**. Receives and writes file using recv() method
- Uses TCP (Tranmission Control Protocol) for reliable transmission

PYTHON SOCKET PROGRAMMING

2. PROJECT COMPONENTS – PYTHON FILES FOR COMMUNICATION

Socket-based file transfer systems are typically broken down into modular Python scripts, each handling a specific part of the communication pipeline.

COMPONENTS OVERVIEW:

Client

- push_client.py
- client_uploader.py
- file_to_send.txt

Server

- push_server.py
- multi_client_server.py
- file_to_send.txt

CLIENT FILES

push_client.py
Connects to the server and initiates the file transfer.

client_uploader.py
Uploads selected files to the server using sockets

file_to_send.txt
A sample file prepared to be sent to the server

SERVER FILES

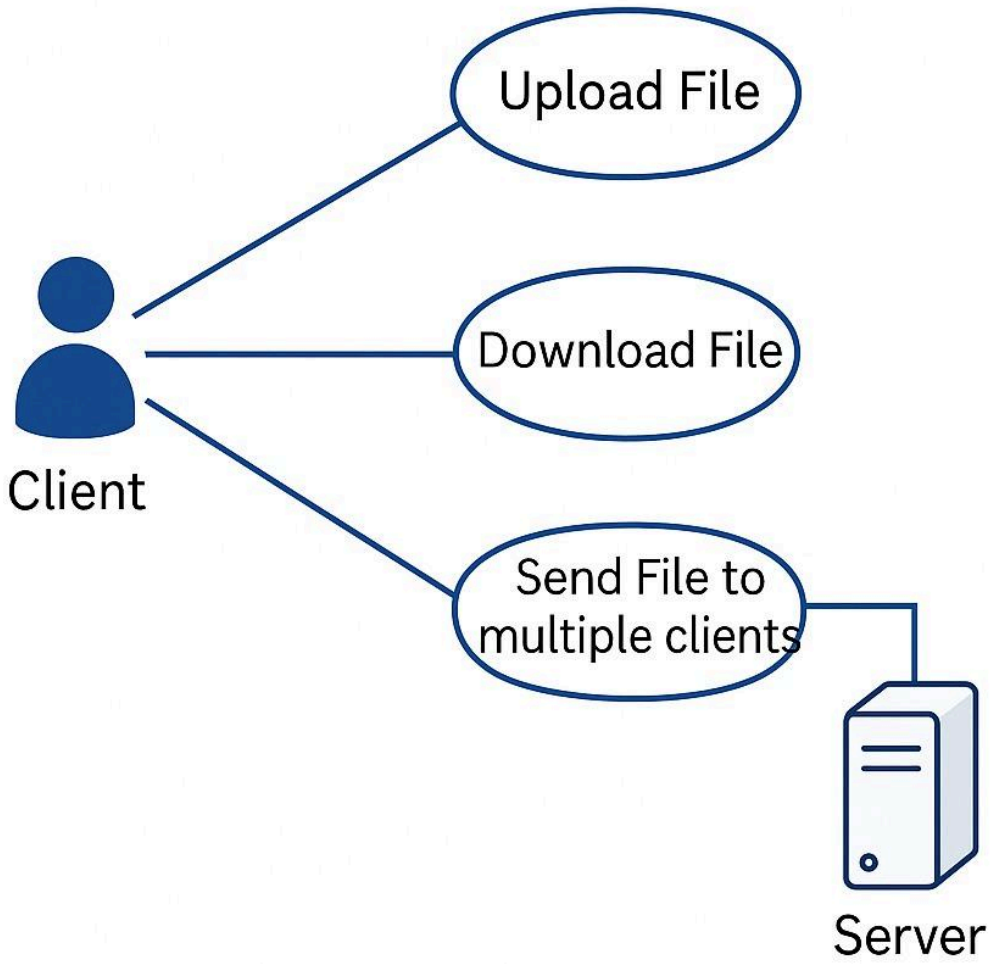
push_server.py
Waits for client connections, receives incoming files, and stores them.

multi_client_server.py
A threaded socket server that handles multiple clients sending files concurrentily

file_to_send.txt
A sample file located on the server side (may be used for sending back or testing)

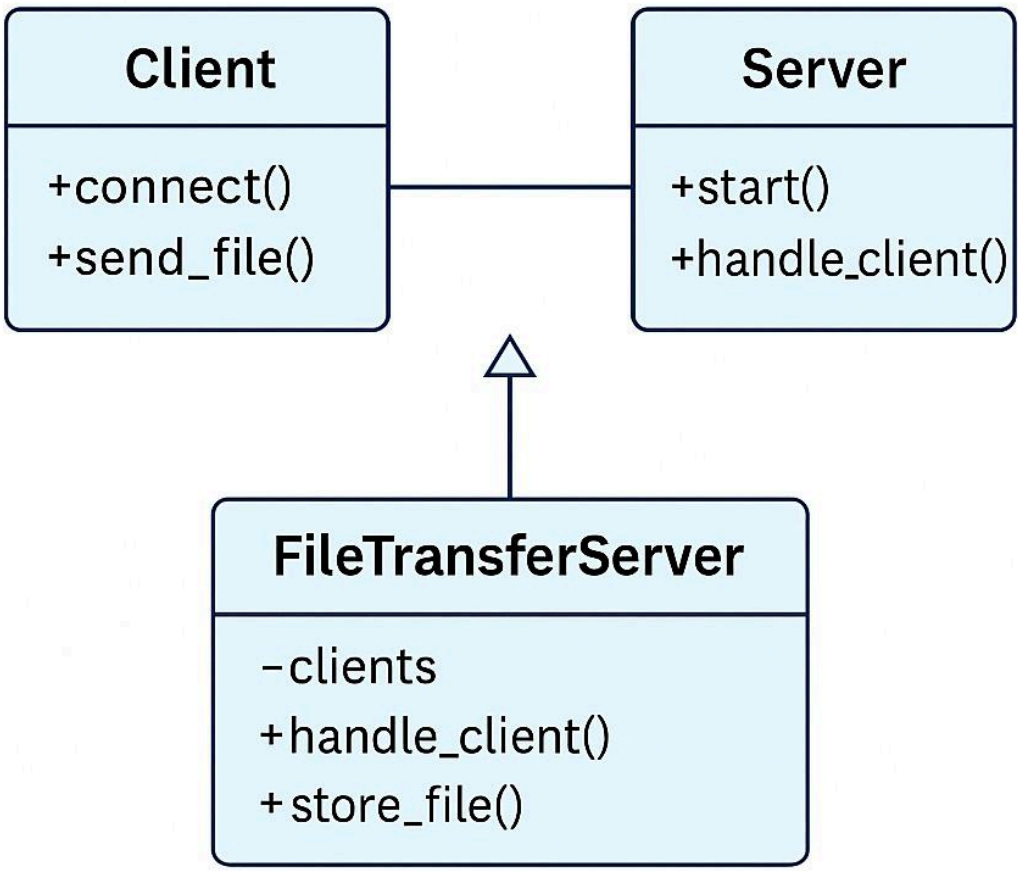
received_55425_file to_send.txt
A file received from a client, used for verification of tranfer

USE CASE DIAGRAM



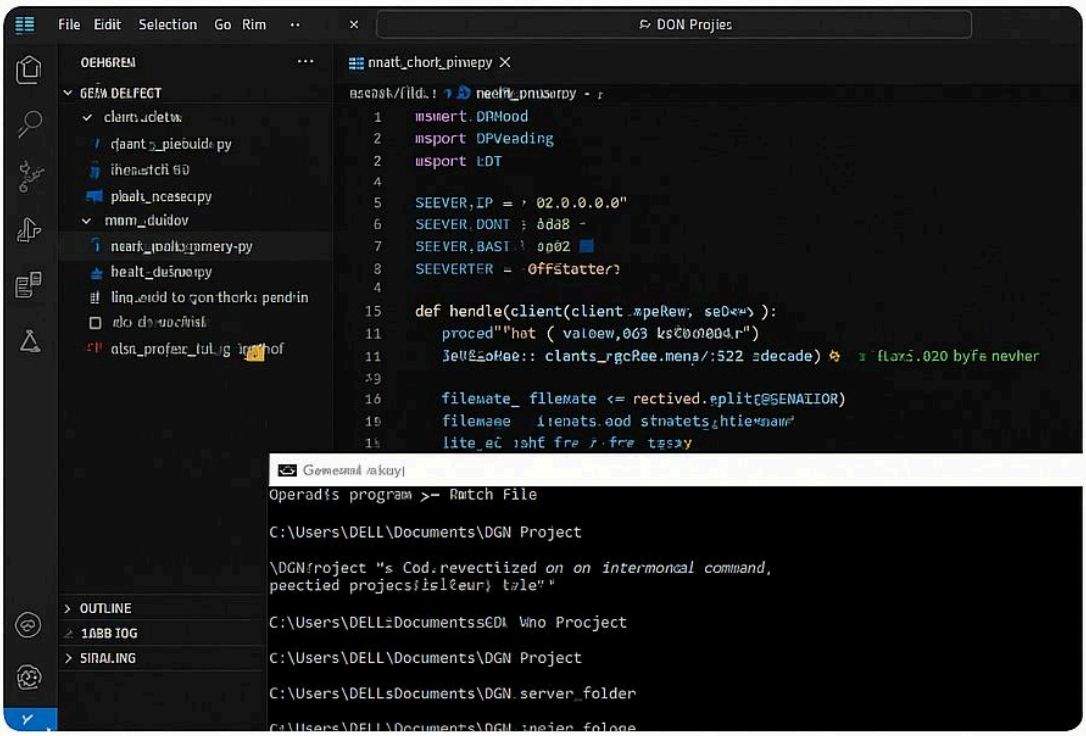
A use case diagram representing the file transfer system. The client can upload and downloud files, while the server can distribute a file to multiple clients.

CLASS DIAGRAM



A class diagram illustrates the structure of the file transfer system implemented in Python using socket programming, including the Client, Server, and FileTransferServer classes.

HOW TO RUN THE PROJECT



1 Open the project folder in VS Code

2 Launch the Terminal in VS Code

3 Run the server script: **python push_server.py**

4 Run the client script: **python push_client.py**

Comnand Prompt

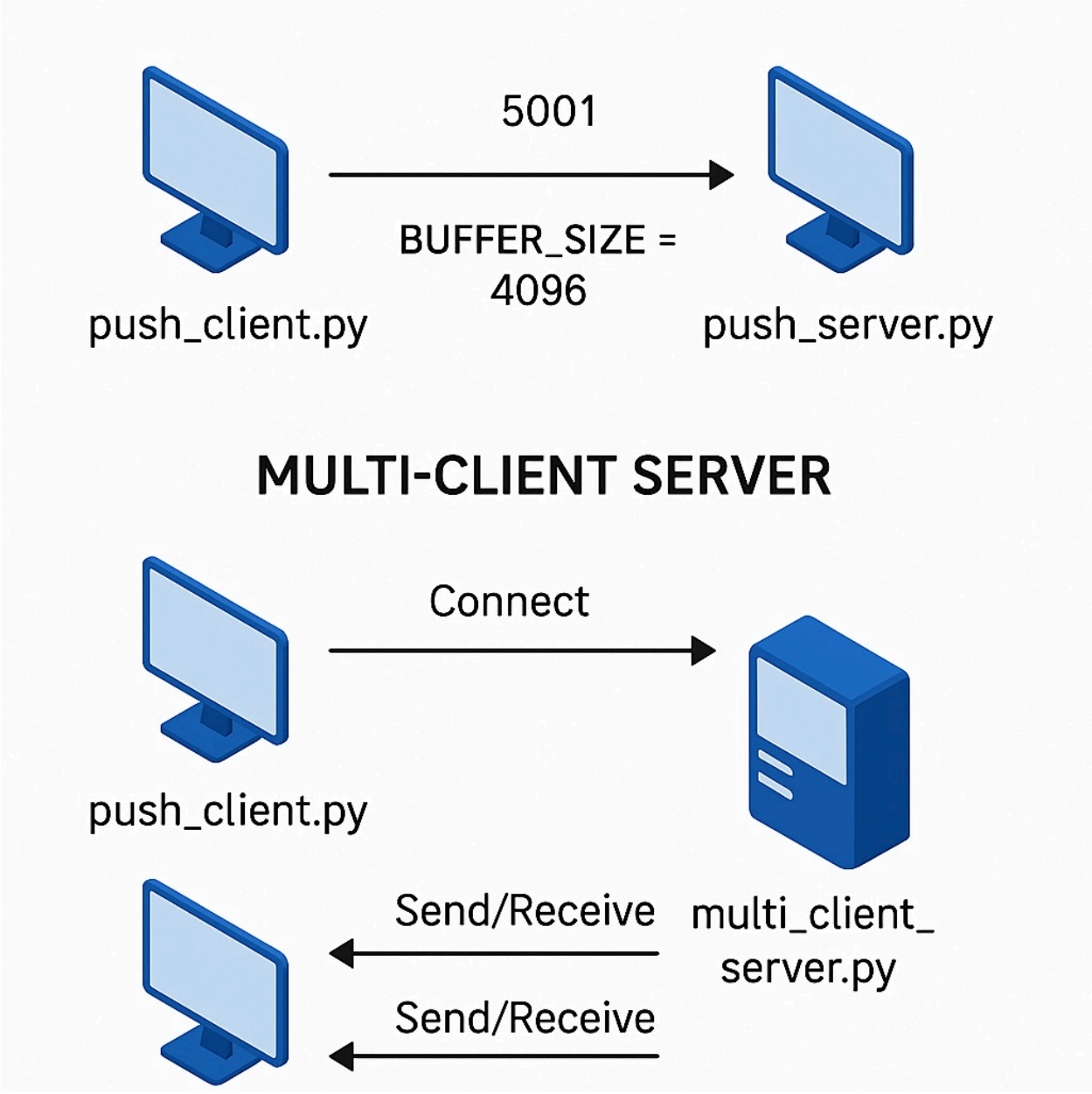
```
operable program or ǰatch ∓ıtė

C:\Users\DELL\Documents\DCN Project

C:\Users\DELL\Documents>cd DCN Proj

C:\Users\DELL\Documents\DCN Project pcd server_folder
```

Client server Architecture



- The server starts listening for incoming connections on 0.0.0:5001

```
C multi_client_server.py

[+] Server listening on 0.0.0:5001
```

- The client uploads file_to_send.txt to the server

```
> flClint_uploader.py>'⌐'

[+] File file_to_send.txt sent success
```

- The server receives and attempts to parse the file header

```
C multi_client_server.py

[+] Server listening on 0.0.0.0:5001
[+] '192.160.100.19",/ connected
[+] Error parsing header from <'193,%0ti to server:
file_to_send.txt <SCRARATOR•24hehehe hehe heheh he
```

FUNCTIONALITIES EXPLAINED

1 FILE UPLOAD BY CLIENT

Client side initiates the connection, selects a file, and uploads it using socket programming.

2 FILE RECEIVING BY SERVER

Server listens for incoming connections, receives file data in chunks, and stores it locally.

3 MULTI-CLIENT SUPPORT

Threaded or asynchronous server architecture to handle multiple clieInts simultaneously

4 ERROR HANDLING / DATA INTEGRITY

Implements error handling to manage failed transfers and checksums to ensure data validity.

LIMITATIONS

Single-Threaded Server

The basic server design handles one file transfer at a time. It cannot accommodate multiple concurrent clients → need multithreading

Lack of Encryption

File data is sent in plain text, making it vulnerable to interception by unauthorized parties → need secure protocols like SSL/TLS

No Interrupt Handling

Currently, the server cannot gracefully handle client disconnections or unexpected errors → need better interrupt handling

No File Size Limit

There is no built-in limit for file uploads, which could lead to excessive memory or storage use → need to implement size checks

FUTURE ENHANCEMENTS

1 ERROR HANDLING

Implement more robust error handing to deal with network failures and file transfer errors.

2 MULTIPLE CLIENTS

Improve the server to handle concurrent connections from multiple clients more efficiently

3 PACKET SIZE

Optimize the packet size used for data transfer to enhance efficiency

4 SECURITY

Add encryption and authentication to secure file transfers

CONCLUSION

A socket-based file transfer system was successfully developed using Python socket programming.

Crafting such a system enhanced our understanding of network communication, socket programming, and multi-client architectures.

This project showcases a practical application of computer networks in facilitating reliable data transfer.