LABWORK 3: MPI

Student's name: Nguyen Trung Kien
Student's ID: BI12-224

1. System architecture

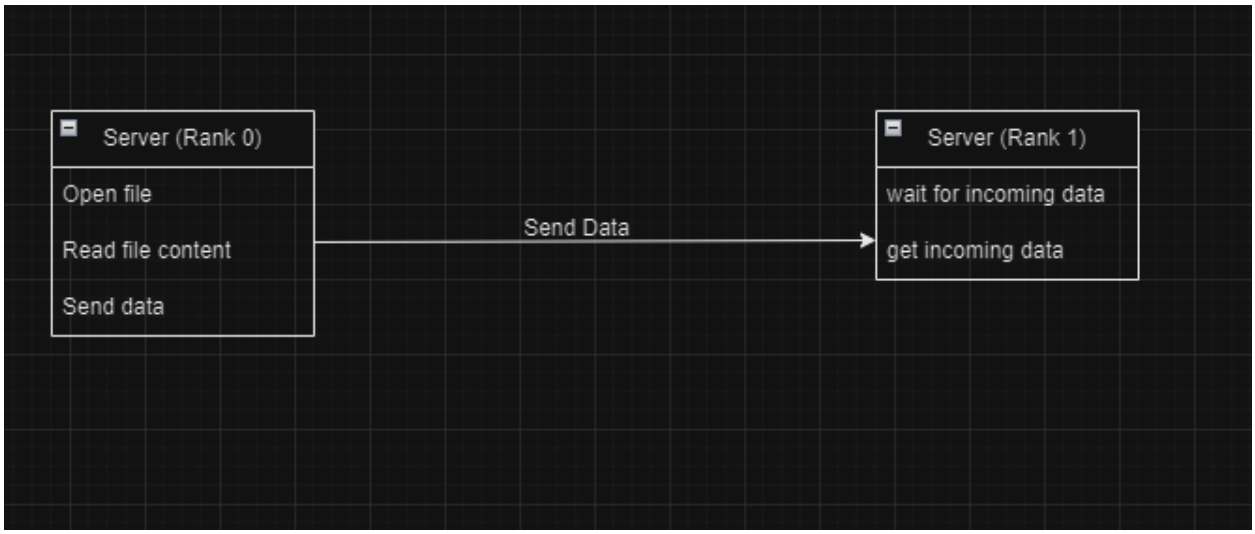
a. Code explain

- It initializes the MPI environment using MPI_Init.
- It retrieves the rank and size of the MPI communicator MPI_COMM_WORLD.
- It checks if there are exactly two processes. If not, it prints an error message and finalizes MPI execution.
- The processes with rank 0 and rank 1 perform different tasks:
    i. Rank 0 (Client):
        1. Opens the file "sent_file.txt" for reading.
        2. Reads the entire content of the file into the buffer.
        3. Sends the buffer containing the file data to the server (process with rank 1) using MPI_Send.
    ii. Rank 1 (Server):
        1. Uses MPI_Probe to check for an incoming message from the client.
        2. Retrieves the count of characters in the message using MPI_Get_count.
        3. Receives the file data sent by the client into the buffer using MPI_Recv.
        4. Opens a file "received_file.txt" for writing.
        5. Writes the received data from the buffer into the file.
- After completing their tasks, both processes finalize the MPI environment using MPI_Finalize.

b. Workflow

- The client process (rank 0) reads the content of "sent_file.txt" into a buffer and sends it to the server process (rank 1).

- The server process receives the data from the client, writes it to "received_file.txt", and acknowledges successful reception.
- The communication between client and server is achieved using MPI point-to-point communication primitives MPI_Send and MPI_Recv.
- The file transfer is one-way (client to server), and there's no error handling for cases such as file not found or communication failure, which could be added for robustness.



2. Implementation

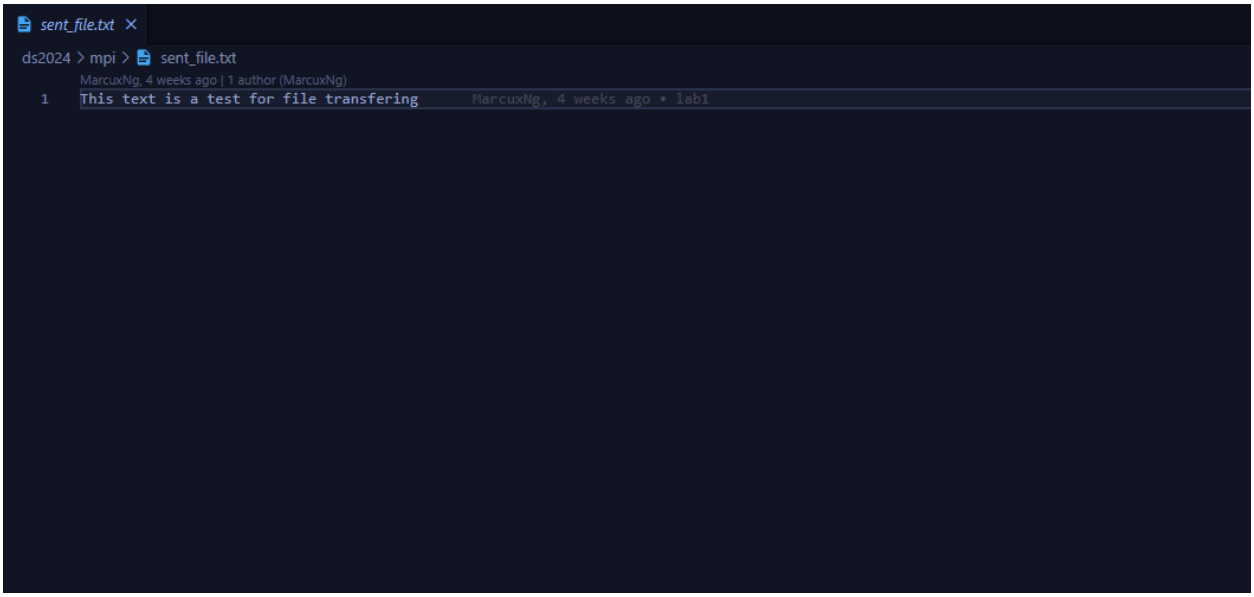


Figure 1. sent_filetxt

```
➜  lab_3 git:(main) X mpicc file_transfer.c -o file_transfer
➜  lab_3 git:(main) X mpirun -np 2 ./file_transfer
[Client] File sent successfully
[Server] Data received and written to 'received_file.txt' successfully
```