LABWORK 3: MPI File Transfer

Student's name: Hoang Quoc Minh Quan
Student's ID: BI12-363

I. Choice of OpenMPI implementation

I choose OpenMPI for my labwork because of its open-source, high performance and Compatibility.

II. System Architecture

The system contains 2 processes: Client(rank 1) and Server(rank 0). The client sends files to the server through a buffer with a specific tag.

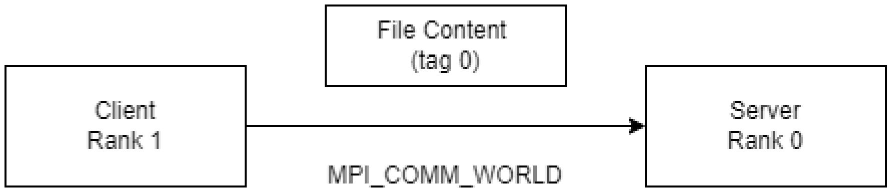
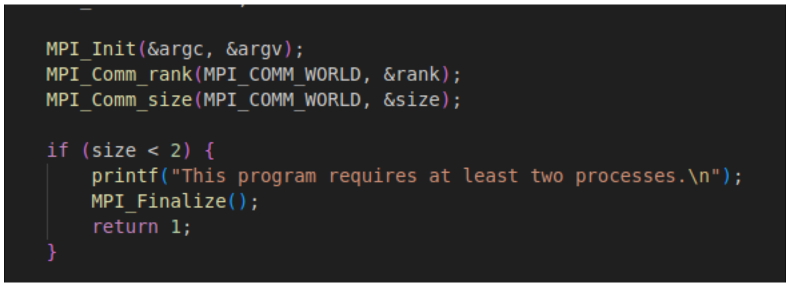


Figure 1.System Architecture

III. Implementation

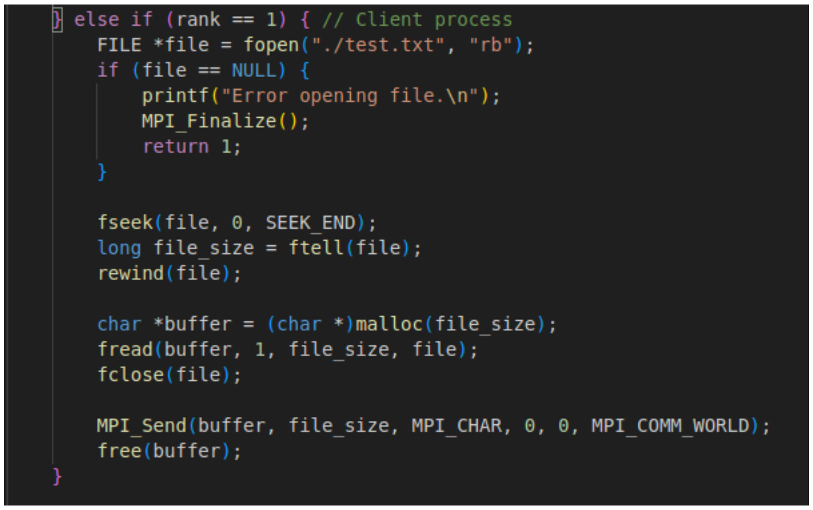
Firstly, the program initializes MPI, retrieves the rank and size of the MPI communicator (MPI_COMM_WORLD). Then, if the number of processes (size) is less than 2, it prints a message and finalizes MPI.



```c
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

if (size < 2) {
    printf("This program requires at least two processes.\n");
    MPI_Finalize();
    return 1;
}
```

In the client process (rank 1), the program opens "test.txt", determines its size, reads the content into a buffer, and sends it to the server (rank 0) using MPI_Send. After sending, it frees the buffer. If the file opening fails, it prints an error message and finalizes MPI.



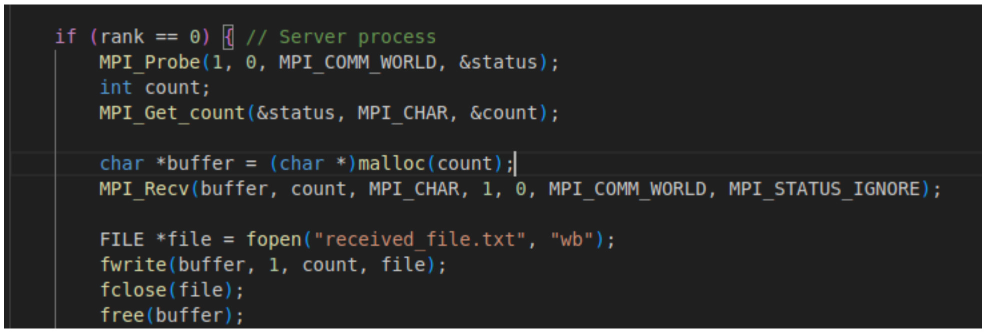
```c
} else if (rank == 1) { // Client process
    FILE *file = fopen("./test.txt", "rb");
    if (file == NULL) {
        printf("Error opening file.\n");
        MPI_Finalize();
        return 1;
    }

    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    rewind(file);

    char *buffer = (char *)malloc(file_size);
    fread(buffer, 1, file_size, file);
    fclose(file);

    MPI_Send(buffer, file_size, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    free(buffer);
}
```

In the server process (rank 0), the program waits for a message from the client (rank 1) to determine its size using MPI_Probe, allocates a buffer based on this size, receives the message using MPI_Recv, and writes the received message to "received_file.txt" using fwrite. It then closes the file and frees the buffer for resource cleanup.



```c
if (rank == 0) { // Server process
    MPI_Probe(1, 0, MPI_COMM_WORLD, &status);
    int count;
    MPI_Get_count(&status, MPI_CHAR, &count);

    char *buffer = (char *)malloc(count);
    MPI_Recv(buffer, count, MPI_CHAR, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    FILE *file = fopen("received_file.txt", "wb");
    fwrite(buffer, 1, count, file);
    fclose(file);
    free(buffer);
```