LABWORK 3: MPI File Transfer

Student's name: Nguyễn Thái Dương

Student's ID: BI12-117

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 0) and Server(rank 1). The client sends files to the server through a buffer with a specific tag.

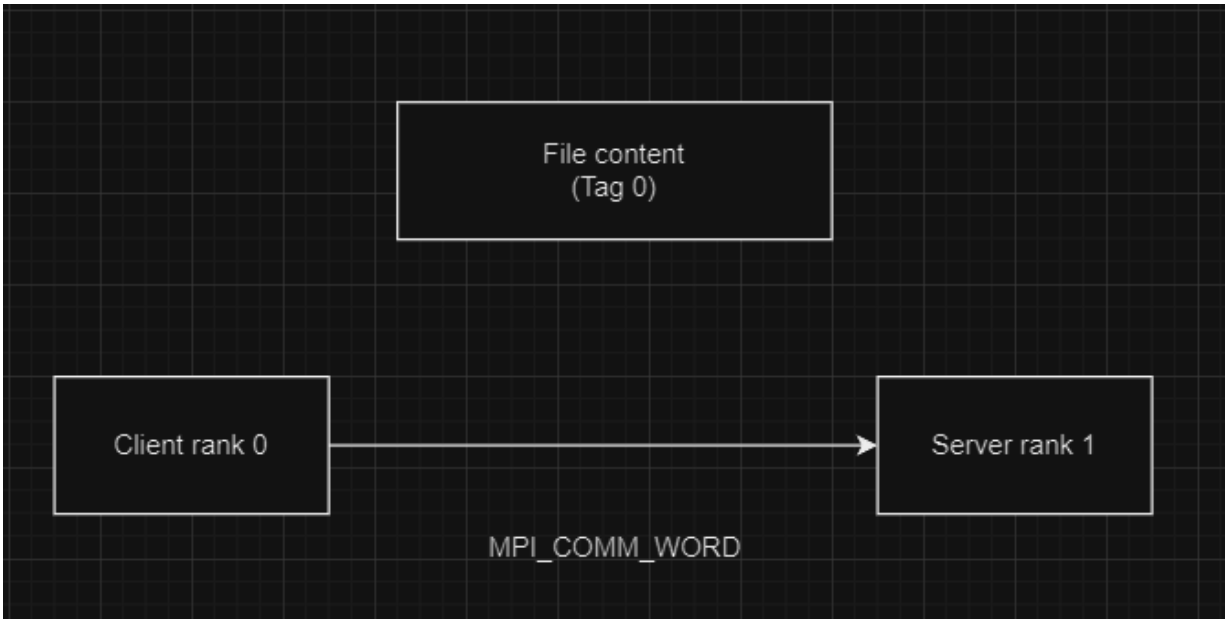

Figure 1.System Architecture

III. Implementation

The program starts by initializing MPI, obtaining the rank and size of the MPI communicator (MPI_COMM_WORLD). If the number of processes (size) is less than 2, it displays a message and concludes MPI operations.

```c
int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size != 2) {
        fprintf(stderr, "This application requires exactly 2 processes.\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
```

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

```c
if (rank == 0) {
    // Open file to send
    FILE *file_to_send = fopen(FILE_NAME, "r");
    if (file_to_send == NULL) {
        perror("File opening failed");
        MPI_Finalize();
        exit(EXIT_FAILURE);
    }

    // Get file size
    fseek(file_to_send, 0, SEEK_END);
    long file_size = ftell(file_to_send);
    fseek(file_to_send, 0, SEEK_SET);

    // Send file size
    MPI_Send(&file_size, 1, MPI_LONG, 1, 0, MPI_COMM_WORLD);

    // Send file data
    char buffer[BUFFER_SIZE];
    size_t bytes_read;
    while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, file_to_send)) > 0) {
        MPI_Send(buffer, bytes_read, MPI_BYTE, 1, 0, MPI_COMM_WORLD);
    }

    // Close file
    fclose(file_to_send);
```

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

```
} else if (rank == 1) {
    // Receive file size
    long file_size;
    MPI_Recv(&file_size, 1, MPI_LONG, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    // Open file to write
    FILE *received_file = fopen("received_file.txt", "w");
    if (received_file == NULL) {
        perror("File opening failed");
        MPI_Finalize();
        exit(EXIT_FAILURE);
    }

    // Receive file data
    char buffer[BUFFER_SIZE];
    size_t remaining_bytes = file_size;
    while (remaining_bytes > 0) {
        size_t bytes_to_receive = (remaining_bytes < BUFFER_SIZE) ? remaining_bytes : BUFFER_SIZE;
        MPI_Recv(buffer, bytes_to_receive, MPI_BYTE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        fwrite(buffer, 1, bytes_to_receive, received_file);
        remaining_bytes -= bytes_to_receive;
    }
```