UNIVERSITY OF SCIENCE & TECHNOLOGY OF HANOI
TRƯỜNG ĐẠI HỌC KHOA HỌC & CÔNG NGHỆ HÀ NỘI

Practical Work 2: MPI

Vũ Đức Hiếu - BI12-162

I) System architecture

II) Implementation

1) The program initializes MPI and retrieves the rank and size of the current process and the total number of processes.
2) It checks if the number of processes is exactly 2. If not, it prints an error message and exits.
3) The client process (rank 0) opens the file `test.txt`, reads its content into the buffer, and sends it to the server process (rank 1).
4) The server process waits for incoming data from the client using MPI_Probe, determines the size of the incoming message, receives the data, and writes it to a new file `received_file.txt`.
5) Finally, MPI is finalized, and the program exits

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define SIZE 1024*10

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

   int rank, size;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Get the rank of the
current process
   MPI_Comm_size(MPI_COMM_WORLD, &size); // Get the total number
of processes

   if (size != 2) {
      printf("This program requires exactly 2 processes (1 client and 1
server).\n");
      MPI_Finalize(); // Finalize MPI
      return 1;
```

```
    }

    char buffer[SIZE]; // Buffer to hold file data
    FILE *file;

    if (rank == 0) { // Client process
        int server_rank = 1; // Rank of the server process

        file = fopen("test.txt", "r"); // Open the file to be sent
        if (file == NULL) {
            printf("[-] error opening file\n");
            MPI_Finalize();
            return 1;
        }

        fseek(file, 0, SEEK_END); // Move file pointer to the end of the file
        long file_size = ftell(file); // Get the size of the file
        fseek(file, 0, SEEK_SET); // Move file pointer back to the beginning of
    the file

        fread(buffer, 1, file_size, file); // Read file content into buffer
        fclose(file); // Close the file

        MPI_Send(buffer, file_size, MPI_CHAR, server_rank, 0,
    MPI_COMM_WORLD); // Send file data to the server
        printf("[Client] File sent successfully\n");

    } else if (rank == 1) { // Server process
        MPI_Status status;
        MPI_Probe(0, 0, MPI_COMM_WORLD, &status); // Probe for incoming
    message from client
        int count;
        MPI_Get_count(&status, MPI_CHAR, &count); // Get the count of
    characters in the incoming message

        MPI_Recv(buffer, count, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE); // Receive file data from client

        file = fopen("received_file.txt", "w"); // Open file to write received data

        fwrite(buffer, 1, count, file); // Write received data to file
        fclose(file); // Close the file

        printf("[Server] Data received and written to 'received_file.txt'
    successfully\n");
    }

    MPI_Finalize(); // Finalize MPI
    return 0;
}
```