Practical Work 3: MPI File Transfer

Vo Truong Giang - BI12-130

December 4, 2025

1 Introduction

In this practical work, I implemented a file transfer system using the Message Passing Interface (MPI). Unlike the previous Client-Server models (TCP/RPC), MPI allows processes to communicate within a parallel computing environment.

2 MPI Implementation Choice

I chose **OpenMPI** because it is the standard, open-source implementation of MPI on Linux/Ubuntu systems. It provides robust tools like `mpicc` for compilation and `mpirun` for execution.

3 System Design

The system runs as a single executable but behaves differently based on the process rank:

- **Rank 0 (Receiver):** Acts as the server. It waits for messages with `TAG_FILENAME` then `TAG_CONTENT`.
- **Rank 1 (Sender):** Acts as the client. It reads the file and sends data to Rank 0.

```
RANK 1 (Sender)                                RANK 0 (Receiver)
   |                                              |
   |--- MPI_Send(Filename, TAG_FILENAME) ---------->| MPI_Recv
   |                                              |
   |--- MPI_Send(Data Chunk, TAG_CONTENT) --------->| MPI_Recv (Loop)
   |--- MPI_Send(Data Chunk, TAG_CONTENT) --------->| MPI_Recv
   |                 ...                          |
   |--- MPI_Send(0 bytes, TAG_CONTENT) ------------>| MPI_Recv (Stop)
   |                                              |
```

Figure 1: MPI Point-to-Point Communication Design

4 Implementation Details

```
// Main logic based on Rank
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

if (world_rank == 0) {
    run_server(); // Uses MPI_Recv loop
} else if (world_rank == 1) {
    run_client(argv[1]); // Uses MPI_Send loop
}
```

5 Conclusion

The MPI implementation successfully transfers files between two processes using blocking point-to-point communication routines.