Practical Work 3 - MPI File Transfer

Tran Vu Cong Minh - 23BI14303

2025-12-15

Goal of the practical work

The primary objective of this laboratory session is to implement a mechanism for **one-to-one file transfer** utilizing the Message Passing Interface (MPI). Unlike the approaches taken in previous practical works—such as low-level sockets (PW1) or Remote Procedure Calls (PW2)—this implementation relies on explicit message passing between distinct MPI processes to exchange data.

The specific goals were to:

- Install and configure the necessary MPI environment.
- Architect a basic server-client model defined by MPI process ranks.
- Execute a file transfer between two distinct MPI nodes.
- Evaluate and compare this MPI-based approach against previous TCP and RPC implementations.

Environment setup

The experiments were conducted within the same Kali Linux virtual machine environment established during earlier sessions.

OpenMPI installation

The OpenMPI library was installed and verified using the following commands:

```
sudo apt update
sudo apt install openmpi-bin libopenmpi-dev
mpicc -v
mpirun --version
```

MPI design for file transfer

Rank-based architecture

The system architecture adopts a server-client model distinguished by MPI ranks:

- **Rank 0 (Server)**: Responsible for reading the source file and transmitting both the file size and its content.
- **Rank 1 (Client)**: Responsible for receiving the metadata (size) and content, and subsequently writing the data to an output file.

The application is executed using `mpirun`, invoking two processes:

```
mpirun -np 2 ./mpi_file_transfer input_file output_file
```

Message format

The communication protocol involves a strictly ordered exchange:

- Rank 0 first transmits the file size as an integer.
- Rank 0 then transmits the actual file payload as `MPI_BYTE`.
- Rank 1 waits to receive the size, allocates the necessary memory buffer, receives the payload, and commits the data to disk.

System organization and Implementation

The entire implementation is encapsulated within a single C source file named `mpi_file_transfer.c`.

The program flow begins by initializing the MPI environment, validating command-line arguments, and determining the rank of the current process. Based on the rank, the program diverges into either server or client logic.

Server Logic (Rank 0)

- Opens and reads the specified input file.
- Calculates the total size of the file.
- Dispatches the file size followed by the file content to Rank 1.

Client Logic (Rank 1)

- Listens for the incoming file size message.
- Listens for the incoming data stream.
- Writes the received data to the specified output path.

Build commands

The source code was compiled using the MPI C compiler wrapper:

```
mpicc -Wall -g mpi_file_transfer.c -o mpi_file_transfer
```

A typical execution command to test the system is as follows:

```
mpirun -np 2 ./mpi_file_transfer example.txt result.txt
```

Execution and test results

To verify functionality, a test file containing a simple string was created:

```
echo "Hello from MPI server" > example.txt
```

Upon running the MPI program, the standard output confirmed the transmission flow:

```
[Sender] Read 22 bytes from 'example.txt', transmitting to receiver...
[Sender] Transmission completed successfully.
[Receiver] Successfully received 22 bytes and saved to 'result.txt'
```

A final check comparing the input and output files confirmed that the transfer was accurate and complete.

Discussion

Comparison with TCP Sockets

- MPI abstracts the complexity of manual socket creation and connection management.
- Communication is addressed logically by **process rank**, rather than by specific IP addresses and port numbers.

Comparison with RPC

- MPI relies on **explicit message passing** operations (Send/Recv).
- While simpler for this specific task, MPI does not provide the automatic data marshalling and interface definition languages found in RPC frameworks like gRPC.

Although MPI is traditionally favored for high-performance parallel computing, it proved to be a viable and efficient solution for this distributed file transfer exercise.

Conclusion

This practical work successfully realized a file transfer system using MPI. It serves as a concluding comparison in our exploration of distributed systems, highlighting the differences between direct socket manipulation, structured Remote Procedure Calls, and MPI-based message passing.