Practical Work 3: MPI File Transfer

Nguyen Thanh Dat
Student ID: 23BI14091

1 Introduction

The goal of this practical work is to implement a file transfer system using the **Message Passing Interface (MPI)** standard. Unlike the previous client-server models (TCP and RPC) where distinct programs connected over a network socket, this system uses a **Single Program, Multiple Data (SPMD)** model where multiple processes communicate via messages.

2 MPI Implementation Choice

For this implementation, I chose **OpenMPI** running on **Kali Linux**.

2.1 Justification

- **Native Support:** Kali Linux (Debian-based) provides native support for OpenMPI via the APT package manager, ensuring a stable environment without the driver compatibility issues often found on Windows.
- **Environment Integration:** As a security-focused distribution, Kali allows for seamless network process management, making it an ideal environment for testing distributed protocols.
- **Library:** I used `mpi4py` because it provides Pythonic bindings for the MPI standard, allowing for high-level object serialization (pickling) while maintaining the underlying performance of the C-based OpenMPI drivers.

3 Service Design

The system consists of two processes launched simultaneously by the MPI executor. They are distinguished by their **Rank**.

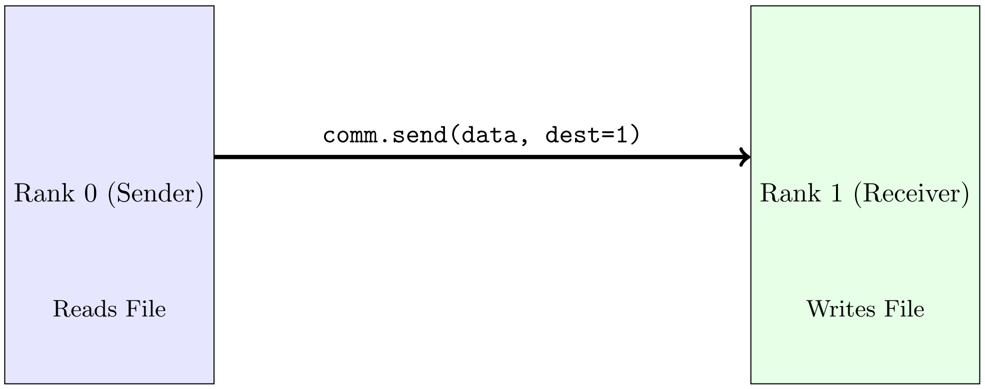


Figure 1: MPI Point-to-Point Communication Design

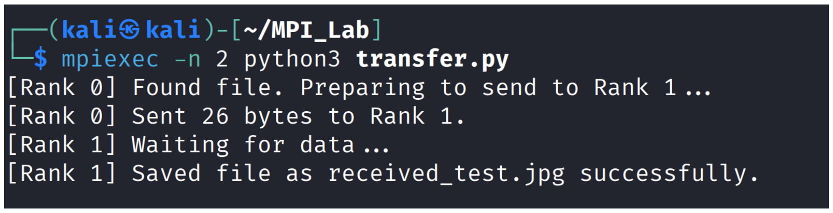
4 Implementation

The logic is contained within a single file, `transfer.py`. The execution path splits based on the process rank.

```
from mpi4py import MPI
import os

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    # SENDER LOGIC
    with open('test.jpg', 'rb') as f:
        file_data = f.read()

    # Package data into a dictionary
    data_package = {'filename': 'test.jpg', 'content':
       file_data}

    # Blocking send
    comm.send(data_package, dest=1, tag=0)
    print(f"Sent {len(file_data)} bytes to Rank 1.")

elif rank == 1:
    # RECEIVER LOGIC
    data = comm.recv(source=0, tag=0)

    with open('received_' + data['filename'], 'wb') as f:
        f.write(data['content'])
    print("File saved successfully.")
```

5 Verification

The following screenshot demonstrates the execution in the Kali Linux terminal. It shows Rank 0 successfully sending the bytes and Rank 1 confirming the save operation.



```
┌──(kali㉿kali)-[~/MPI_Lab]
└─$ mpiexec -n 2 python3 transfer.py
[Rank 0] Found file. Preparing to send to Rank 1...
[Rank 0] Sent 26 bytes to Rank 1.
[Rank 1] Waiting for data...
[Rank 1] Saved file as received_test.jpg successfully.
```

Figure 2: Successful MPI Execution on Kali Linux