MPI File Transfer System - Short Report

1. Why I Chose My MPI Implementation

I selected the mpi4py implementation because it provides a simple Python interface to the standard MPI API while retaining full compatibility with widely used MPI implementations such as OpenMPI and MPICH. This allows rapid prototyping, easy integration with Python-based tools, and cross-platform operability without modifying low-level C-based MPI code.

2. Design of the MPI Service

```
+-------------------+        MPI_Send / MPI_Recv        +---------------------+
|   Sender (Rank 0) | --------------------------------> | Receiver (Rank 1)   |
| - Reads file      |                                     | - Writes file     |
| - Chunks to bytes | <--------------------------------  | - Confirms receipt |
+-------------------+         Acknowledgement            +---------------------+
```

3. System Organization

```
System Structure:
-----------------
MPI/
■■■ mpi_file_transfer.py
■   ■■ Rank 0: File sender
■   ■■ Rank 1: File receiver
■■■ README (optional)

Execution Model:
mpiexec -n 2 python mpi_file_transfer.py
 - Rank 0 prompts user for file input
 - Rank 1 receives and stores file
```

4. Code Snippet: File Transfer Implementation

```
# Sending a chunk of binary data
with open(filepath, "rb") as f:
    while sent < filesize:
        data = f.read(CHUNK)
        arr = np.frombuffer(data, dtype=np.uint8)
        comm.Send(arr, dest=1, tag=2)

# Receiving a chunk
buffer = np.empty(min(CHUNK, filesize - received), dtype=np.uint8)
comm.Recv(buffer, source=0, tag=2)
f.write(buffer.tobytes())
```