MPI File Transfer System

This report details the creation of a file transfer system using the Message Passing Interface (MPI) standard, specifically implemented with Python's `mpi4py` library. This approach moves away from the traditional Client/Server or RPC models and utilizes a Peer-to-Peer message passing paradigm suitable for parallel computing, where two distinct processes (Rank 0 and Rank 1) coordinate the transfer.

MPI File Transfer Design

1. **Design Rationale:** Unlike RPC, MPI does not inherently use Client/Server roles. Instead, it assigns **Ranks** to define the task of each parallel process within the `MPI_COMM_WORLD` communicator.
    - **Rank 0 (Sender):** Reads the file from the disk and is responsible for sending the metadata and subsequent data chunks.
    - **Rank 1 (Receiver):** Receives the data and reassembles the file onto the disk.
2. **Communication Protocol:** The system relies on Point-to-Point communication routines (`comm.send` and `comm.recv`), distinguished by a **Tag** value to manage the flow of information. The communication is executed in a sequential strategy:
    - **Step 1 (Tag 1):** Transfer **Metadata** (Filename, Filesize).
    - **Step 2 (Tag 2):** Transfer **Data Chunks** (Binary Bytes).
    - **Step 3 (Tag 2):** Send a **Termination Signal** (Empty bytes) to indicate the end of the file.

System Organization and Execution

The entire process is contained within a single program (`mpi_transfer.py`), which is launched by the MPI runtime environment. The command **`mpiexec -n 2`** spawns two instances of the program that communicate over a channel abstracted by the MPI standard.

- **Process 0 (Sender):** Executes the sender function (`run_sender`), using explicit `comm.send()` calls to push the metadata and data packets to Process 1.
- **Process 1 (Receiver):** Executes the receiver function (`run_receiver`), utilizing explicit `comm.recv()` calls to poll for incoming messages with specific tags and write them to the designated output file.

Conclusion

The MPI implementation successfully achieved file transfer using a parallel, peer-to-peer communication model. This architecture is notably different from both TCP (low-level explicit protocol) and RPC (high-level function abstraction) as it requires explicit communication calls but gains from the highly optimized data handling inherent to the MPI standard. The core mechanism relies on differentiating roles based on the **rank** and using **tag** values to enforce the correct sequence of communication.