More Hints

When designing your protocol the following points re. the use of pipes may be helpful:

- By default (you do not specify `O_NONBLOCK`) pipes are self-synchronized. This means that if you attempt to read from a pipe which has no data (but still has at least one process having it open for writing), then the read will block. Similarly, if you attempt to write to a pipe which is full, then the write will block.

- There is no kind of push-back or unget for data read from a pipe. This means that once a process reads even a single byte of data from a pipe then there is no way that some other process can re-read that data.

- Pipes provide a byte stream, they do not maintain any boundaries between packets or structures. If boundaries are relevant, then they must be maintained at a higher level by the protocol which uses the pipes.

- The simplest pipe protocols would dedicate a single pipe for one-way communication between every pair of processes. If you design your protocol carefully, then it is possible to use a single pipe with:

    - Multiple writer processess and a single reader process. In this case, you need to ensure that:

        - The reader can somehow distinguish data boundaries.

        - The reader can identify the writer process or does not care about the identity of the writer process.

        - If you write fewer than `PIPE_BUF` bytes in a **single** write and read those bytes using a **single** read then you are guaranteed to receive all those bytes correctly without that data getting interspersed with data from other writers. POSIX requires that `PIPE_BUF` is at least 512 bytes.

          This means that you may write a small fixed number of integers in a single write and are guaranteed to be able to read them with a single read. However, there is no guarantee that you can transmit an arbitrarily sized matrix or even a single row/column of an arbitrarily sized matrix without getting that data interspersed with other writes.

    - Multiple reader processes with a single writer process. In this case, you need to ensure that:

        - The reader processes can somehow identify data boundaries.

        - The protocol does not care about which reader process receives the written data.

        - If a protocol depends on sending variable-sized data by having a writer writing an initial count followed by that number of data entities, then a reader process would first need to issue a read to get the initial count followed by **another** read to get the data entities. However, this would not work as another reader may read from the pipe during the two read's.

- Multiple writer processes with multiple reader processes. The considerations for both of the previous cases apply.