Unix domain sockets IPC (flow-controlled)

Flow of data implemented as descirbed. Note that the program output is verbose enough to testify the flow. To run, use `make`, further instructions will be on your screen.

- SOCKET: Very standard implementation using calls to `socket()`, `bind()`, `listen()`, `accept()` in P1 for setting up and `connect()` inside P2 for communication. The socket uses `AF_UNIX`. Standard `read`, `write` calls are used to send and receive data. `sscanf` parses the data by matching regex of the *5 strings* sent by P1 to P2. I have sent the 5 strings in the format described below (it remains consistent throughout).

  ```
  [index].    stringOf8Bytes
  [index+1]   stringOf8Bytes
  ...
  [index+4]   stringOf8Bytes
  ```

  This way, just matching the first digit suffices. All of the communications are in `BLOCKING` mode, i.e. a process will wait for data to be written in the socket. Incase a packet fails or the correct acknowledgement is not sent, the sender will resend that packet (this is consistent throughout). Finally, after the said objective is accomplished, the server is unlinked and both processes quit.

- FIFO : Nothing were special here too. FIFO's/named pipes are bidirectional and `BLOCKING` by default. Here, a fifo file writes data only when there is a process reading it, otherwise the write calls are blocked. This allows us to conveniently send and receive data with the help of the same pipe. Similar thing also happens in the sockets. The rest implementation details are same as above.

- Message Queues (POSIX): Not a fan of this one. POSIX message queues do not allow convenient two way comms. However SYSTEMV message queues do have a `msgtyp` attribute to distinguish messages. I have created two message queues. One for writing data FOR process P2 and another for writing data FOR process P1 (i.e. the ACK). Message queues are received by priority, however this question doesn't really need that, so all priorities are set to 0 thereby making it almost a FIFO. The rest details remain same.