There are 3 folders,

1) **FIFO's**
2) **Sockets**
3) **Message Queues**

Each of these folders contains p1, p2 and makefile to run the programs. Some folders also include other necessary .txt files (don't delete).

To compile and to get executables p1 and p2 do 'make'.
To see the generated output and working of the programs run ./p2 first in one terminal and ./p1 in another terminal.

Basic Working:

In all the p1 programs a function "randString" is defined which is used to generate 50 random strings.
Structure "mesg_buffer" is the data structure that is used to store the data. Index of the string and the string itself this structure is used to pass the data between processes.

```
struct mesg_buffer {
    int index;
    char mesg_text[100];
} dataset[50];
```

FIFO's :

- int mkfifo(const char *pathname, mode_t mode);
- mkfifo() makes a FIFO special file with name pathname.
- FIFO has to be opened at both ends to make it work.
- As named FIFO is kind of a file, all the read, write, close, open system calls can be used with it, and this is used to communicate between p1 and p2.

- Here in p1, after generating 50 random strings and storing their indexes in the mesg_Buffer structure they are passed to the FIFO by a write system call and p2 using read system call reads all the strings with their index. Now p1 waits till it receives the string back with the highest index among the strings sent earlier from p2. This process goes on till all 50 strings are passed.

- Error handling:

  For all the system calls used namely write, read, close, open, whenever any error occurs errno is set accordingly. These errors are printed using perror().

MESSAGE QUEUE:

Here two message queue's are used in p1 one to send the data (struct mesg_buffer)
Which contains the string and its index and the other to receive one string (with the highest index)
Also, the same message queues are made in p2 where one of them receives the strings sent by p1 and the other one sends one particular string which has the highest index among the 5 sent by p1.
This string is received by p1 and it is printed.
Mesg_buffer is sent in a batch of 5 by p1 and then it waits till it receives the string with the highest index sent by p2.
Sending of mesg_buffer is done by

```
msgsnd(msgid_1, &dataset[0], sizeof(dataset[0]), 0);
```

.Receiving of mesg_buffer is dony by:

```
msgrcv(msgid_2, &message, sizeof(message), 1, 0);
```

msgget gets a System V message queue identifier.
When passed a key, and proper flags.
Two keys are generated using ftok(), which creates two message queue's identifiers.

- **Error handling:**

  For all the system calls used and queue related API's used, whenever any error occurs errno is set accordingly. These errors are printed using perror().

SOCKETS:

In p1 and p2 a socket is created which returns a file descriptor.

```
data_socket = socket(AF_UNIX, SOCK_SEQPACKET, 0);
```

- **AF_UNIX is used for local communication.**
- **For protocol 0 is passed which chooses the default protocol.**
- **SOCK_SEQPACKET provides a sequenced, reliable two-way communication.**

- P1 acts firstly as a client and uses the following API's socket(), connect(), send(), recv(), and close().

- P2 acts as a server and uses the following API's socket(), bind(), listen(), accept() recv(), close() and send().

- socket() creates a new communication endpoint, so p1 and p2 can communicate.
- bind() associates and reserves the port for use by the socket
- listen() is to announce willingness to accept any connections
- accept() is used to block the caller until a connection request arrives
- send() to send data(here mesg_buffer) over the connection established.
- recv() to receive data(here mesg_buffer) over the connection established.
- connect() it attempts to establish a connection.
- close() to release the connection.

p1, after generating 50 random strings and storing their indexes in the mesg_Buffer structure they are passed to p2. Now, p1 waits till it receives the string back with the highest index among the strings sent earlier from p2. This process goes on till all 50 strings are passed between the p1 and p2.

- **Error handling:**
  For all the system calls and other API's used, whenever any error occurs errno is set accordingly. These errors are printed using perror().