Assignment 3

Question 2 Documentation

Program Logic / How the Program Works:

1. **Sockets:**
   I have made two P1_S.c and P2_S.c files to create a server-client setup.
   In P1_S, I first made a random string generator function and then I made an array of length 50 and added 50 random strings into the randomStrings collection.
   Then I made a socket of the AF_UNIX family and then bind and listened to the socket. Then I took a file descriptor fp which I opened using the open syscall.
   Then I start a loop that runs until the ID received from P2_S of the random string sent is less than 50.
   In the loop, I send a batch of 5 strings from the current ID value to P2_S using the send function and then using the read function, the ID is received from P2_S, and then the loop continues.
   In P2_S, I make the socket with the same configs and use the connect function to connect to P1_S and in a loop open the file descriptor to read data from P1_S. After reading the data, I use the ID of the last batch string and send it to P1_S to read.
   DATA STRUCTURE: Sockets provide point-to-point, two-way communication between two processes. Sockets are very versatile and essential components of interprocess and intersystem communication. A socket is an endpoint of communication to which a name can be bound. We can use sockets to communicate between processes on a single system, like other forms of IPC. The UNIX domain (AF_UNIX) provides a socket address space on a single system. UNIX domain sockets are named with UNIX paths.

2. **FIFO:**
   Here again, we have P1_F.c and P2_F.c files for making a server-client setup.
   In P1_F, I make a FIFO file in the tmp folder using the mkfifo function. Then same as the sockets part, I make an array of 50 random strings.
   Now I start a loop that runs until the ID of the random string provided by P2_F is less than 50.
   In the loop, I open the FIFO using the open call in O_WRONLY mode and then write a batch of 5 random strings starting from the current value of ID, into the FIFO file and then close the fd.
   Then I again open the FIFO in O_RDONLY to read the ID supplied by P2_F.
   In P2_F, I again declare the FIFO in tmp folder and start an infinite loop in where I open the FIFO in O_RDONLY mode to first read the strings supplied by P1_F in the FIFO and then in O_WRONLY mode to write the ID of the last string received on the FIFO.
   DATA STRUCTURE: A *FIFO special file* is similar to a pipe, but instead of being an anonymous, temporary connection, a FIFO has a name as any other file. Processes open the FIFO by name in order to communicate through it. A pipe or FIFO has to be open at both ends simultaneously. If you read from a pipe or FIFO file that doesn't have any processes writing to it (perhaps because they have all closed the file, or exited), the read returns end-of-file.

3. **MESSAGE PASSING QUEUES:**

We have two P1_Q.c and P2_Q.c files for making a server-client setup.
In P1_Q, we make a message buffer queue using struct and then make two pointers of this messageBuffer struct.
Then as done in the above parts I make an array consisting of 50 random strings and then start a loop for sending and receiving data.
The loop runs until the ID of the random string provided by P2_Q is less than 50. P1_Q first sends a batch of 5 random strings by using the messageText pointer and concatenating the 5 strings into this pointer and then using the msgsend function to send the pointer to P2_Q using the msgid which is generated by the msgget function and the struct object.
After sending the batch of strings P1_Q receives the ID of the last string sent using the msgrcv function.
In P2_Q, I again make the same struct and the 2 objects of the struct and then a loop starts in which the msgids are generated using msgget and the data is received using msgrcv and printed and then the ID of the last string received is sent using the msgsend function to P1_Q.
DATA STRUCTURE: Message Passing IPC messaging lets processes send and receive messages, and queue messages for processing in an arbitrary order. Unlike the file byte-stream data flow of pipes, each IPC message has an explicit length. Messages can be assigned a specific type. Because of this, a server process can direct message traffic between clients on its queue by using the client process PID as the message type.

<u>How to run the Program :</u>

First, come into the Q2 directory
To compile all the 6 files P1_S, P2_S, P1_F, P2_F, P1_Q and P2_Q in one go run:

```
make
```

To compile only the Socket files namely P1_S and P2_S run:

```
make socket
```

To compile only the FIFO files namely P1_F and P2_F run:

```
make fifo
```

To compile only the message passing queue files namely P1_Q and P2_Q run:

```
make msgqueue
```

After compilation of the files first, always run the executable of P1 (./P1_X) and then in a separate parallel terminal run the P2(./P2_X) executable to see the result.