CSE231 - Operating Systems
Assignment 3
Exercise 3.2

Name: Abhik S Basu
Roll Number: 2020165
Section: A
Branch: CSE

Task: InterProcess Communication using Sockets, Message Queues & FIFO Queues

---

I have submitted 3 folders namely: Message Queue, Sockets and FIFO Queue.

Since the implementation of all 3 of my codes are located in different folders hence there will be 3 separate makes for 3 different methods of IPC. The make commands are as follows:

`make`
To make and compile all the clients and server files

`make clean`
To clean all the executable files and to run the code again as well as to unbind the sockets and to clean the FIFO queues as well

`make unbind`
To unbind the sockets

makefile

```
make umake: c s
c: client.c
    gcc client.c -o client
s: server.c
    gcc server.c -o server
clean:
    rm server client *.socket
unbind:
    rm *.socket
```

makefile

```
make: c s
c: client.c
    gcc client.c -o client

s: server.c
    gcc server.c -o server

clean:
    rm server client fifo1 fifo2
```

makefile

```
make: c s

c: client.c
    gcc client.c -o client

s: server.c
    gcc server.c -o server
clean:
    rm server client
```

To see the code getting executed we need to do the following:

1) Open two terminals and navigate it to the same directory as that of the folders.
2) Run `./server` (`./server &`) first.

3) Run ./client now in the other terminal window.

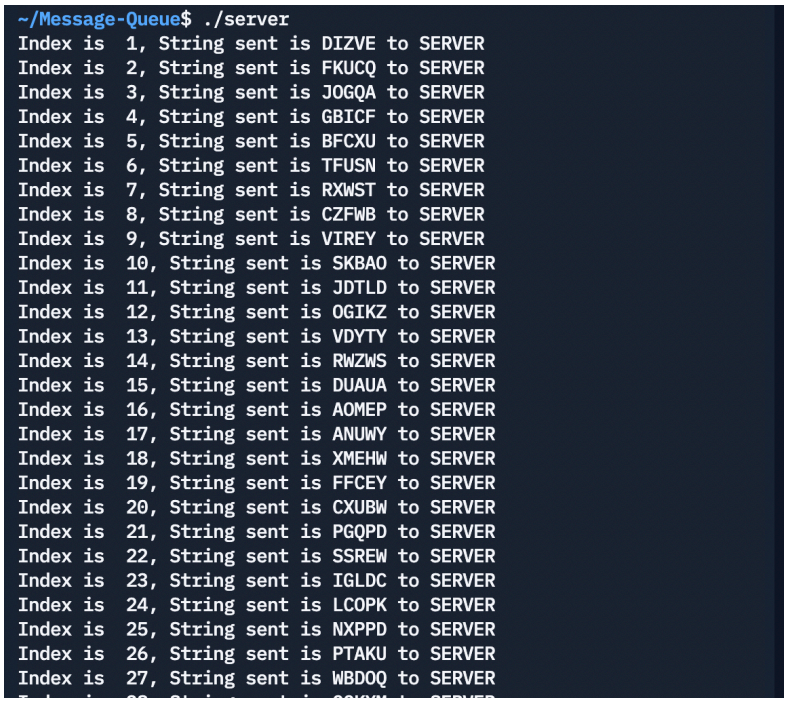
The outputs will be as follows for the Sockets, FIFO Queues and Message queues respectively.

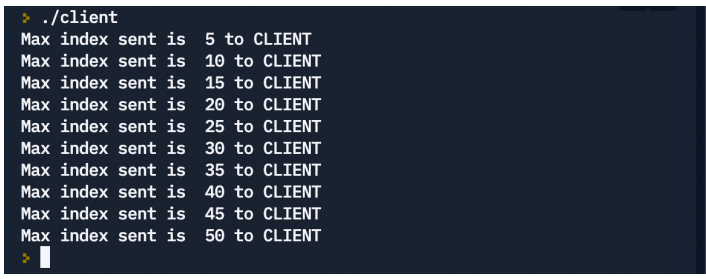
```
~/Sockets$ ./server &
[1] 60
~/Sockets$ Index is  1, String sent is FERKX to SERVER
Index is  2, String sent is BZZRV to SERVER
Index is  3, String sent is VKOXO to SERVER
Index is  4, String sent is UCFQE to SERVER
Index is  5, String sent is KHWQS to SERVER
Index is  6, String sent is FYPNE to SERVER
Index is  7, String sent is AUKTG to SERVER
Index is  8, String sent is JUFJO to SERVER
Index is  9, String sent is DGARD to SERVER
Index is  10, String sent is ONIVE to SERVER
Index is  11, String sent is OHNKX to SERVER
Index is  12, String sent is IRVXF to SERVER
Index is  13, String sent is BAZMT to SERVER
Index is  14, String sent is IVOPG to SERVER
Index is  15, String sent is ESOEK to SERVER
Index is  16, String sent is SUZCQ to SERVER
Index is  17, String sent is FQZTC to SERVER
Index is  18, String sent is XBTUA to SERVER
Index is  19, String sent is AYAAM to SERVER
Index is  20, String sent is WIHMZ to SERVER
Index is  21, String sent is QQSEU to SERVER
Index is  22, String sent is CYOBC to SERVER
Index is  23, String sent is GHUGC to SERVER
Index is  24, String sent is WFFSB to SERVER
Index is  25, String sent is HSZIU to SERVER
```

```
~ ./client
Max index sent is  5 to CLIENT
Max index sent is  10 to CLIENT
Max index sent is  15 to CLIENT
Max index sent is  20 to CLIENT
Max index sent is  25 to CLIENT
Max index sent is  30 to CLIENT
Max index sent is  35 to CLIENT
Max index sent is  40 to CLIENT
Max index sent is  45 to CLIENT
Max index sent is  50 to CLIENT
~
```

```
~/FIFO-Queue$ ./server
Index is  1, String sent is SVNJU to SERVER
Index is  2, String sent is JIGDD to SERVER
Index is  3, String sent is BUZRO to SERVER
Index is  4, String sent is LRDOL to SERVER
Index is  5, String sent is BHJXI to SERVER
Index is  6, String sent is AYERB to SERVER
Index is  7, String sent is XJYMS to SERVER
Index is  8, String sent is SVCYA to SERVER
Index is  9, String sent is FAXET to SERVER
Index is  10, String sent is LPNOD to SERVER
Index is  11, String sent is YSNJR to SERVER
Index is  12, String sent is XJRED to SERVER
Index is  13, String sent is UBMSP to SERVER
Index is  14, String sent is HKNJL to SERVER
Index is  15, String sent is NPLMV to SERVER
Index is  16, String sent is GXLTO to SERVER
Index is  17, String sent is QTGFE to SERVER
Index is  18, String sent is ZDQRH to SERVER
Index is  19, String sent is TNKHG to SERVER
Index is  20, String sent is ZOSMA to SERVER
Index is  21, String sent is DCPQO to SERVER
Index is  22, String sent is MZMXU to SERVER
Index is  23, String sent is CQQKV to SERVER
Index is  24, String sent is UJAMC to SERVER
Index is  25, String sent is JHQTR to SERVER
Index is  26, String sent is WVFQH to SERVER
Index is  27, String sent is HWLYM to SERVER
Index is  28, String sent is CLLQK to SERVER
```

```
~ ./client
Max index sent is  5 to CLIENT
Max index sent is  10 to CLIENT
Max index sent is  15 to CLIENT
Max index sent is  20 to CLIENT
Max index sent is  25 to CLIENT
Max index sent is  30 to CLIENT
Max index sent is  35 to CLIENT
Max index sent is  40 to CLIENT
Max index sent is  45 to CLIENT
Max index sent is  50 to CLIENT
~
```





Now before explaining the logic of the code, first we explain how we have made the random function:

In order to make sure that each time the string generated is random we have made use of the srand function as well so that new random is generated with each iteration. We have further made a 2D array whose first 5 indices will store the different random characters of our string and then we have assigned the string its index on a one indexed basis.

Now we will roughly explain what is happening in each of the methods:

1) Sockets: We use the AF_UNIX socket family in our implementation since we have to carry out the IPC within the same machine. Other than this we make use of the sockaddr_un structure in order to create our socket and use it. We then initialise the attributes of this structure by making use of our socket family. We also make a buffer of 7 which will consist of our 5 random character string as well as our index of string and a newline. Now in the client we make an infinite loop

where we pass the data through the sockets using write() function, five at a time. At the same time in the server.c function we first initialise the socket as done in client.c and then in our first infinite loop we accept the connection socket. The reason we use infinite loops are is to make sure that the programme flow does not terminate abruptly and the process keeps on going till all data is transmitted from client to server and then we close our socket connection only.After accepting the data 5 at a time we assign the result the index of the maximum and then using sprintf to assign this result to buffer we write it on to our socket once again in the server using the write function.Now in the client.c function it just reads the index sent and then prints it. After all packets of data are sent we close the data socket and our client and server terminate. All possible error cases have been handled as well.

2) FIFO Queues: In FIFO queue as well the flow of the programme is more or less similar but some major differences w.r.t sockets are that we first specify the path of our fifo and then make the queue using the mkfifo() function. After that we once again enter into the infinite loop however this time we open the paths using the open() function and specify the functionality of it (whether it is read only or write only). Other than this the flow is more or less similar. We have also made use of the usleep() function in order to make sure that there is a delay and the client does not keep sending data which the server can’t process within the required time. So to prevent any sort of erroneous output or deadlocks or race conditions we have made sure that the programme sleeps at the right time so that it has appropriate amount of time to process the data properly.

3) Message Queues: Once again the flow is similar to the previous two however in this case we make use of two queues and make use of the mesg_buffer structure as well. We first get the id of the two message queues after creating the keys for them using the ftok() function. The once we get the id we get into our infinite loop and once again do the same thing as always. However this time we do it by using msgrcv() and msgsnd() functions of the message queue method. Furthermore, we make sure to destroy the queues once the execution is complete in order to make sure that the queues get closed. Rest of the flow of the programme is similar to the other two. Here we just make sure to properly initialise all attributes of the mesg_buffer struct in the correct manner. All read and write errors have been handled appropriately as well. To check if our queue has been created or not we can make use of the ipcs -q command to see all running queues along with their IDs and their mode.In this programme also a sleep has been put in order to avoid erroneous output and to avoid all potential data hazards.

References:

https://www.geeksforgeeks.org/named-pipe-fifo-example-c-program/
https://www.geeksforgeeks.org/ipc-using-message-queues/?ref=lbp
https://man7.org/linux/man-pages/man7/unix.7.html