Writeup for Q2

Note:- make file will only generate p1 and p2 executable files. One must type the ./p1 and ./p2 commands after make command. Moreover, commands should follow this order

1:- make

2:- ./p1

3:- ./p2

If commands are executed in a different order then the program might not execute.

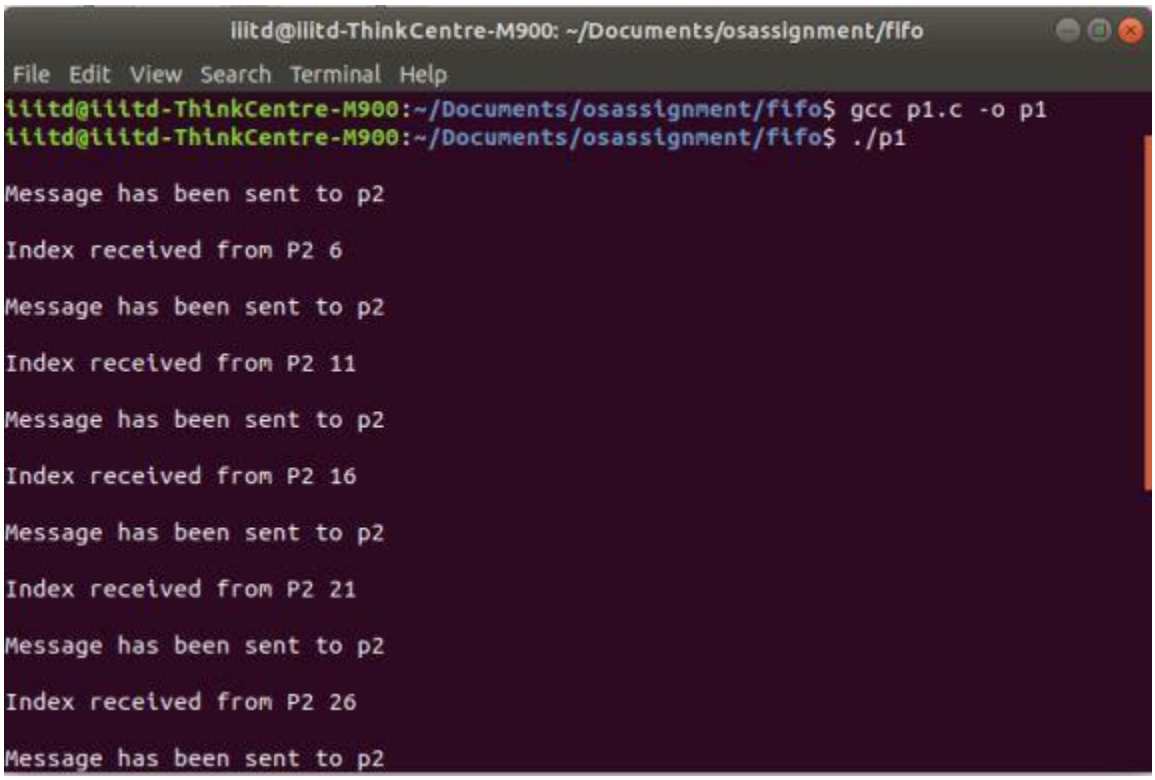
FIFO IPC:-

To establish a two-way communication using FIFO we must use named FIFO with a common file descriptor in the /tmp directory, so that both the process can communicate with each other.

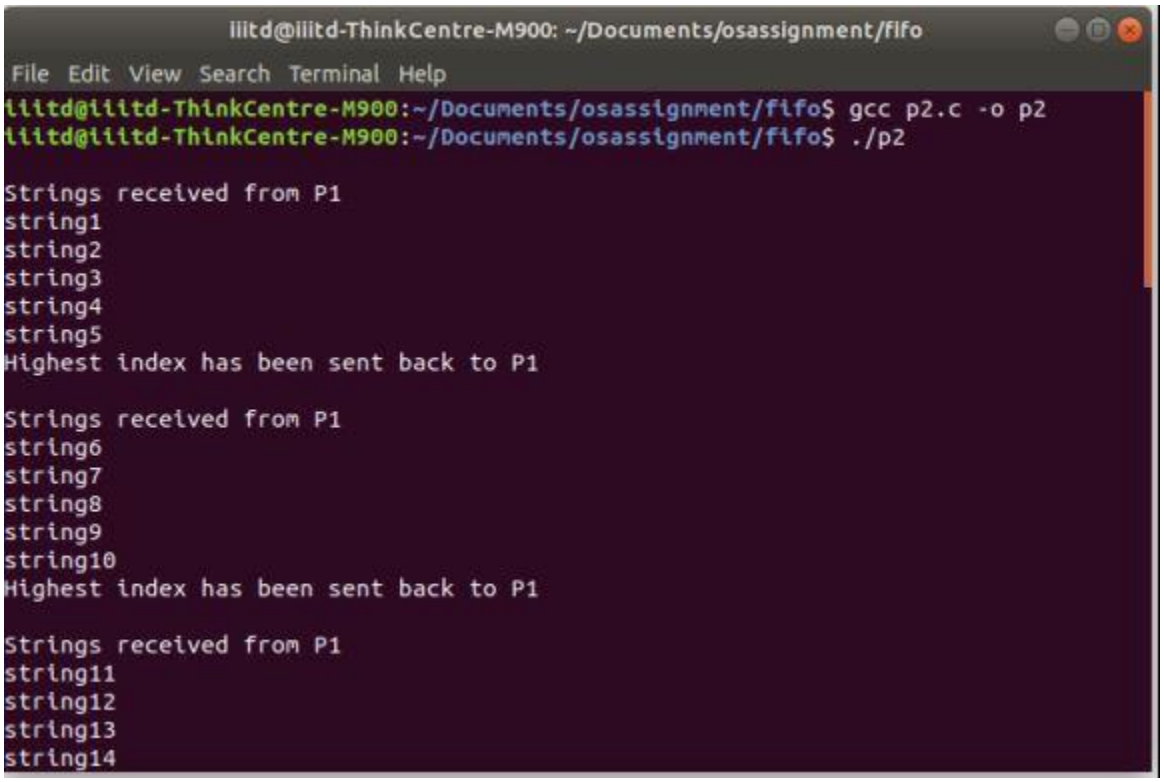
```
int mkfifo(const char *pathname, mode_t mode);
```

mkfifo() system call is used to create a special FIFO file with a specific path and mode. More can be read, write or both. After creating the fifo file we shall open the file using open() system call and use read() and write() to read and write from the file.

Screenshot of the output of the p1:-



Screenshot of the output of the p2:-



In the program p1 is sending a group of strings with the index to p2 and p2 is printing them on the screen and sending the highest index back to p1. This process will be repeated until all the 50 strings are sent to p2.

Messages passing using QUEUE:-

To use the queue as a medium of inter-process communication we shall have to define a structure with two fields.

```
struct message
{
   long type;
   char data[100];
};
```

type is the message type and it is long int. Data is the message that we wish to send to the other process.

To start the communication we must define a common key using ftok() system call. A key is a common file that both the processes will use to communicate.

```
key = ftok("common.c",'b');
```

After defining a common key we shall use msgsend() system call to send the message to another program/process.

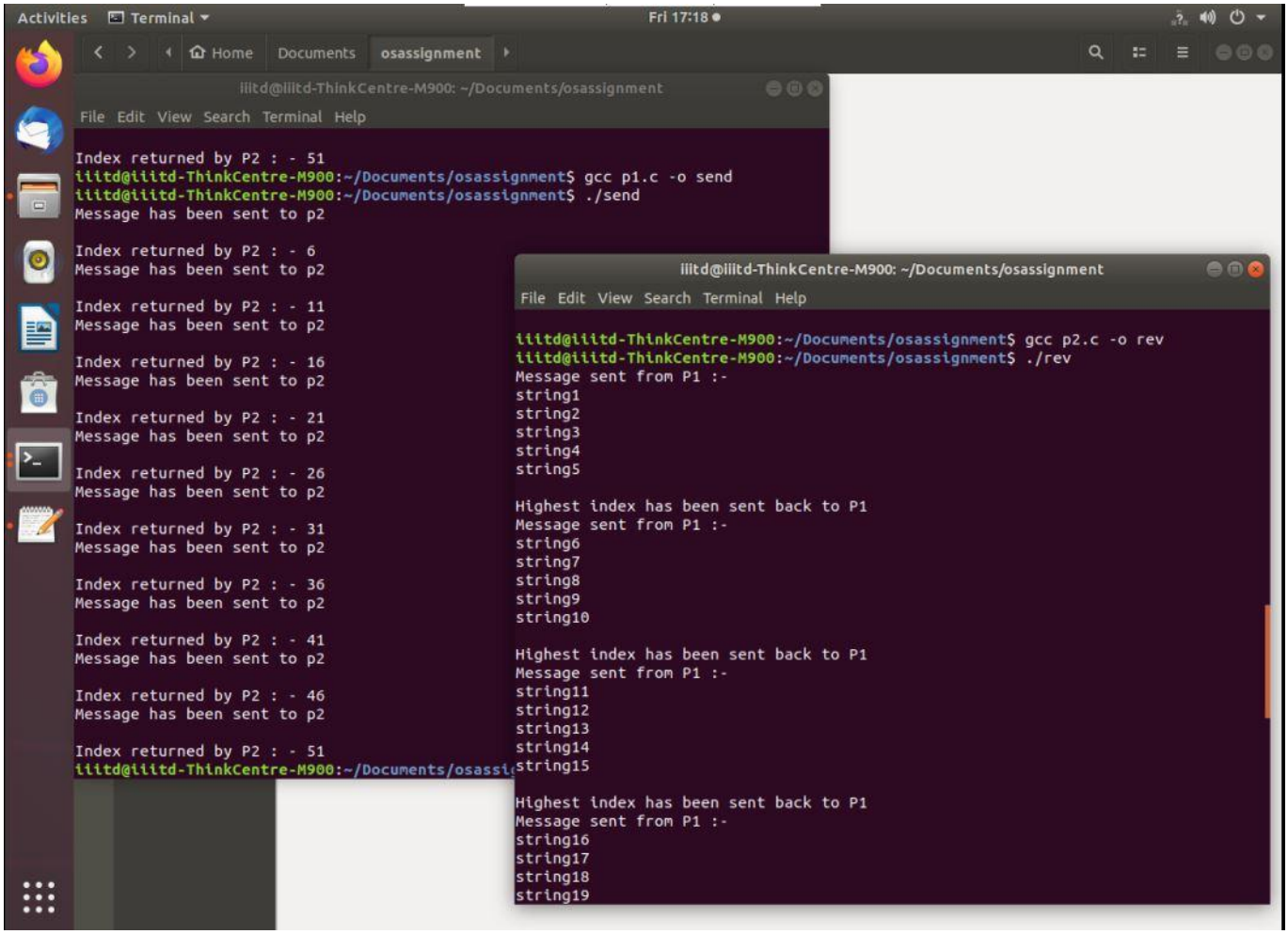
```
msgsnd(msgid, &msg, sizeof(msg), 0)
```

where msg is the instance of the structure.

After sending the message form the p1 we shall receive that message in p2 using msgsnd() system call.

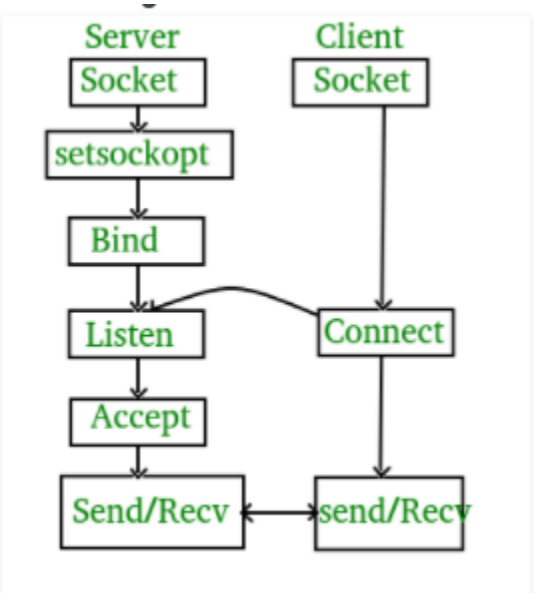
```
msgrcv(msgid,&msg,sizeof(msg),1,0)
```

Screenshot of the output:-



IPC using UNIX DOMAIN SOCKETS

Socket programming is a way to connect two nodes on a network to communicate with each other.
There are many stages to establish the network between the two processes.



First we create a socket using sockfd() system call

```
int sockfd = socket(domain, type, protocol)
```

Then we bind the socket and port number using bind() system call

```
int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);
```

To receive a message we wait for the other process to send something using listen() system call

```
int listen(int sockfd, int backlog);
```

To accept the connection request we use accept() system call

```
int new_socket= accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

To connect the two process to exchange messages we use connect() system call

```
int connect(int sockfd, const struct sockaddr *addr,
                            socklen_t addrlen);
```

Once the connection has been established now we can exchange messages using read() and send() system calls.

<u>Output of the Unix domain socket program:-</u>