Practical-4

Aim:- Implement TWO-WAY Inter-process communication using FIFOs. Consider TWO independent processes for communication. This communication must continue until a specific key is pressed or any process sends a STOP message.

Theory:-
FIFOs are a type of named pipe, which means they have a name associated with them and exist as files on the filesystem. Unlike anonymous pipes created with the pipe() system call, which are used for communication between parent and child processes or between processes created by forking, FIFOs can be used for communication between unrelated processes. FIFOs are bidirectional, allowing data to flow in both directions.
To create a FIFO, you use the mkfifo() system call in C.FIFO operations are blocking, meaning if one process tries to read from an empty FIFO or write to a full FIFO, it will block until the other end is ready.
FIFOs provide a simple and efficient way to implement IPC but may require proper synchronization mechanisms to handle more complex communication scenarios and prevent issues like deadlocks.

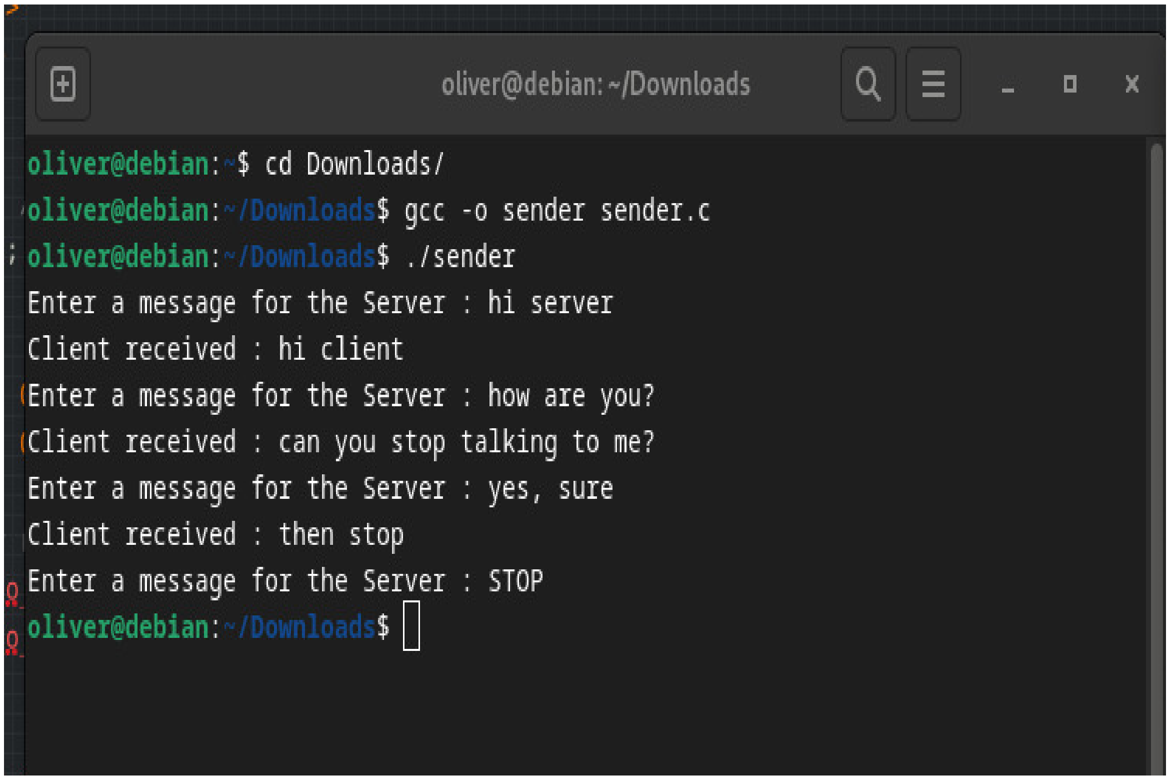
Code:-
Client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main() {
    int fd;
    char message[100];

    mkfifo("fifo", 0666);

    fd = open("fifo", O_RDWR);

    while (1) {
        printf("Enter a message for the Server : ");
        fgets(message, sizeof(message), stdin);

        write(fd, message, strlen(message) + 1);

        if (strncmp(message, "STOP", 4) == 0) {
            break;
        }

        read(fd, message, sizeof(message));
        printf("Client received : %s", message);
    }

    close(fd);
    unlink("fifo");

    return 0;
}
```

Server.c

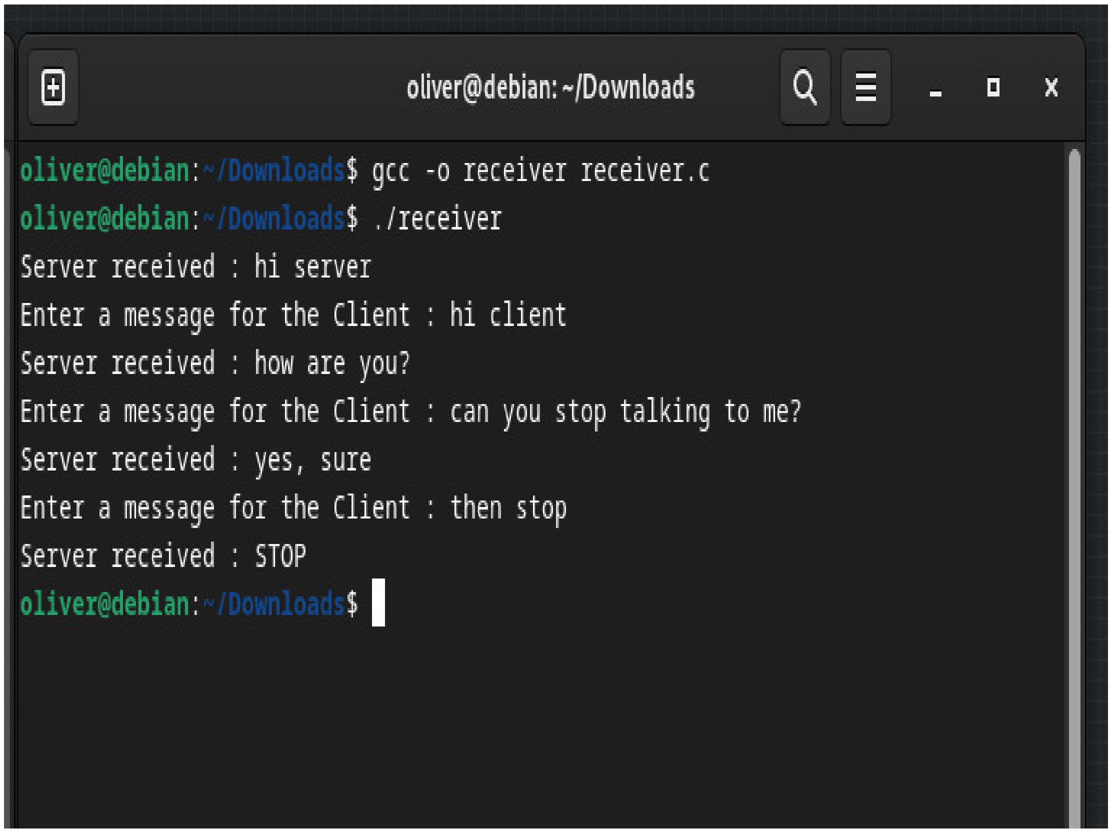
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main() {
    int fd;
    char message[100];

    mkfifo("fifo", 0666);

    fd = open("fifo", O_RDWR);

    while (1) {
        read(fd, message, sizeof(message));
        printf("Server received : %s", message);
        if (strncmp(message, "STOP", 4) == 0) {
            break;
        }

        printf("Enter a message for the Client : ");
        fgets(message, sizeof(message), stdin);

        write(fd, message, strlen(message) + 1);
    }

    close(fd);
    unlink("fifo");

    return 0;
}
```

Output:



oliver@debian: ~/Downloads

```
oliver@debian:~$ cd Downloads/
oliver@debian:~/Downloads$ gcc -o sender sender.c
oliver@debian:~/Downloads$ ./sender
Enter a message for the Server : hi server
Client received : hi client
Enter a message for the Server : how are you?
Client received : can you stop talking to me?
Enter a message for the Server : yes, sure
Client received : then stop
Enter a message for the Server : STOP
oliver@debian:~/Downloads$ 
```



```
oliver@debian:~/Downloads$ gcc -o receiver receiver.c
oliver@debian:~/Downloads$ ./receiver
Server received : hi server
Enter a message for the Client : hi client
Server received : how are you?
Enter a message for the Client : can you stop talking to me?
Server received : yes, sure
Enter a message for the Client : then stop
Server received : STOP
oliver@debian:~/Downloads$
```