FIFO

Aim

Implement FIFO's using IPC channels.

Problem statement:

Write a client - sever program to make the client send the file name and to make the server send back the contents of the requested file implementing it by FIFO's or message queues usong IPC channels.

Description

The program is implemented using a Server-Client model where the server-side is being run on one terminal window and the client-side is run on another terminal window. Initially, server listens to the requests made by the client. Once the client sends a request, the request is processed by the server. Now the server is in the read mode, i.e. it can read the read the request.

The client-side ideally consists of a FIFO structure set-up using the "mkfifo" command. The requests have to be followed by responses from the server-side. Hence, another FIFO structure is used to imlplement the response that gets generated at the server-side. Thus we have two FIFO systems in place, namely request in read mode and response in write mode.

The request consists of the name of the file. Once a request is placed by the client, on the server side, the respective file is opened and contents read. The contents are then passed to the FIFO structure of the client side and thus the contents are displayed.

Algorithm:

Server Side

1. Create two string variables for file name and buffer.

2. Create three integer variables for Request,Response and file operations.

3. Create two FIFO files for response and request using the function mkfifo and pass the respective parameters.

4. Pass the request from the client side as any file to the FIFO.

5. Using file operation variable open the file in read mode

6. Check the file for any characters contained in it. If not present print File not found. else print the contents of the file by passing the contents of the file to the buffer variable using a while loop.

7. Print "request sent".

8. Unlink the response and request file descriptors.

Client Side

1. Create two string variables for filename and buffer.

2. Create two integer variables for Request file descriptor and Response descriptors.

3. Using the request file descriptors open the fifo file which has the request queue and using the response ponter open the fifo file which has he resposne queue in it.

4. Pass the name of the file having contents to the filename variable.

5. Output the contents obtained from the server side.

6. Close the request and response descriptors using closde function.

Code

Server side

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h> /*used for file handling*/
#include <sys/stat.h> /*used for mkfifo function*/
#include <sys/types.h> /*when compiled in gcc, mkfifo() has dependency on both
types.h and stat.h*/

int main()
{
    char fname[50], buffer[1025];
    int req, res, n, file;
    mkfifo("req.fifo", 0777);
    mkfifo("res.fifo", 0777);
    printf("Waiting for request...\n");
    req = open("req.fifo", O_RDONLY);
    res = open("res.fifo", O_WRONLY);
    read(req, fname, sizeof(fname));
    printf("Received request for %s\n", fname);
    file = open(fname, O_RDONLY);
    if (file < 0)
        write(res, "File not found\n", 15);
    else {
```

```c
        while((n = read(file, buffer, sizeof(buffer))) > 0) {
            write(res, buffer, n);
        }
    }
    close(req);
    close(res);
    unlink("req.fifo");
    unlink("res.fifo");
    return 0;
}
```

Client side

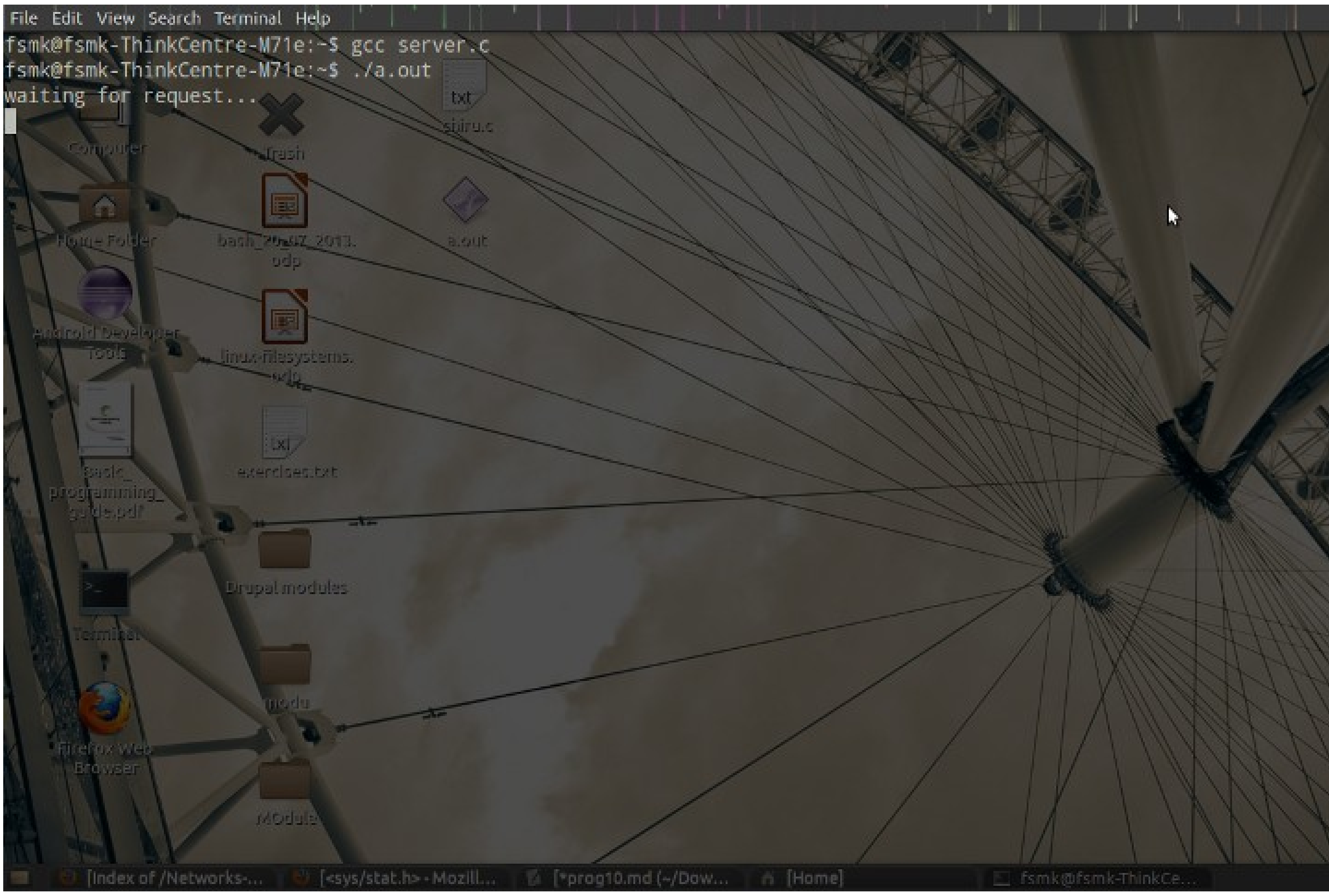
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
    char fname[50], buffer[1025];
    int req, res, n;
    req = open("req.fifo", O_WRONLY);
    res = open("res.fifo", O_RDONLY);
    if(req < 0 || res < 0) {
        printf("Please Start the server first\n");
        exit(-1);
    }
    printf("Enter filename to request:\n");
    scanf("%s", fname);
    write(req, fname, sizeof(fname));
    printf("Received response\n");
    while((n = read(res, buffer, sizeof(buffer)))>0) {
        write(1, buffer, n);
    }
    close(req);
    close(res);
    return 0;
}
```

Output

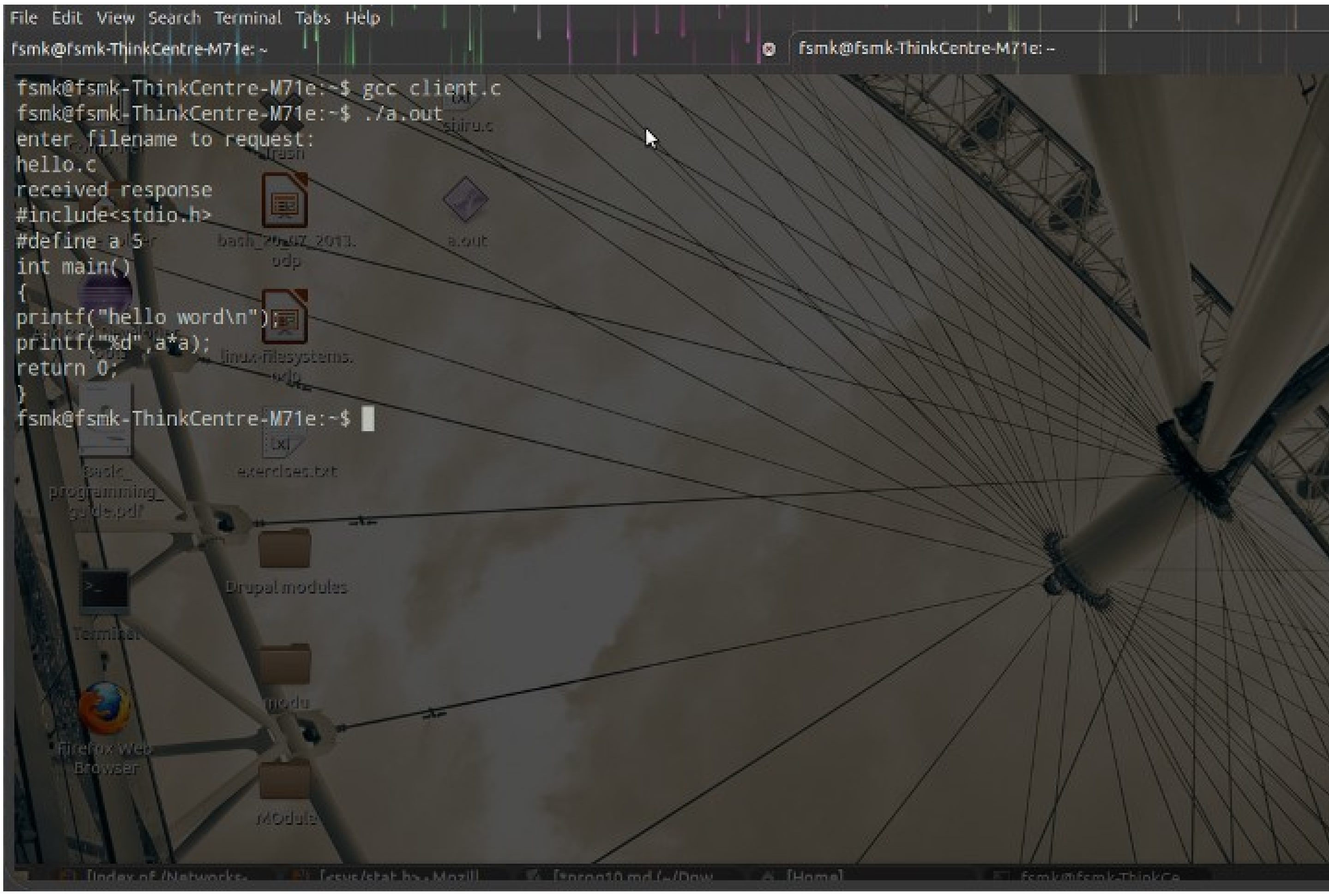
Commands for execution:-

- Open a terminal.
- Change directory to the file location.
- Run g++ filename.cpp
- If there are no errors, run ./a.out

Screenshots:-



```
fsmk@fsmk-ThinkCentre-M71e:~$ gcc server.c
fsmk@fsmk-ThinkCentre-M71e:~$ ./a.out
waiting for request...
```



File Edit View Search Terminal Tabs Help
fsmk@fsmk-ThinkCentre-M71e: ~
fsmk@fsmk-ThinkCentre-M71e: ~

```
fsmk@fsmk-ThinkCentre-M71e:~$ gcc client.c
fsmk@fsmk-ThinkCentre-M71e:~$ ./a.out
enter filename to request:
hello.c
received response
#include<stdio.h>
#define a 5
int main()
{
printf("hello word\n");
printf("%d",a*a);
return 0;
}
fsmk@fsmk-ThinkCentre-M71e:~$ 
```