

## ASSIGNMENT NO: 7-A

**Title:** Inter process communication

**AIM:** Inter process communication in Linux using FIFOs.

### Objectives:

Implementation of Full duplex communication between two independent processes. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays on standard output.

### Theory:

#### FIFOs

A first-in, first-out (FIFO) file is a pipe that has a name in the filesystem. Any process can open or close the FIFO; the processes on either end of the pipe need not be related to each other. FIFOs are also called named pipes

You can make a FIFO using the `mkfifo` command. Specify the path to the FIFO on the command line. For example, create a FIFO in `/tmp/fifo` by invoking this:

```
% mkfifo /tmp/fifo
```

```
% ls -l /tmp/fifo
```

```
prw-rw-rw-
```

```
1 samuel
```

```
users
```

```
0 Jan 16 14:04 /tmp/fifo
```

The first character of the output from `ls` is `p`, indicating that this file is actually a FIFO (named pipe). In one window, read from the FIFO by invoking the following:

```
% cat < /tmp/fifo
```

In a second window, write to the FIFO by invoking this:

```
% cat > /tmp/fifo
```

Then type in some lines of text. Each time you press Enter, the line of text is sent through the FIFO and appears in the first window. Close the FIFO by pressing `Ctrl+D` in the second window. Remove the FIFO with this line:

```
% rm /tmp/fifo
```

## Creating a FIFO

Create a FIFO programmatically using the `mkfifo` function. The first argument is the path at which to create the FIFO; the second parameter specifies the pipe's owner, group, and world permissions, and a pipe must have a reader and a writer, the permissions must include both read and write permissions. If the pipe cannot be created (for instance, if a file with that name already exists), `mkfifo` returns `-1`. Include `<sys/types.h>` and `<sys/stat.h>` if you call `mkfifo`.

## Accessing a FIFO

Access a FIFO just like an ordinary file. To communicate through a FIFO, one program must open it for writing, and another program must open it for reading. Either low-level I/O functions like `open`, `write`, `read`, `close` or C library I/O functions (`fopen`, `fprintf`, `fscanf`, `fclose`, and soon) may be used.

For example, to write a buffer of data to a FIFO using low-level I/O routines, you could use this code:

```
intfd = open (fifo_path, O_WRONLY);  
  
write (fd, data, data_length);  
  
close (fd);
```

To read a string from the FIFO using C library I/O functions, you could use this code:

```
FILE* fifo = fopen (fifo_path, "r");  
  
fscanf (fifo, "%s", buffer);  
  
fclose (fifo);
```

A FIFO can have multiple readers or multiple writers. Bytes from each writer are written atomically up to a maximum size of `PIPE_BUF` (4KB on Linux). Chunks from simultaneous writers can be interleaved. Similar rules apply to simultaneous reads.

## PROGRAM - PIPELINE 1 :

```
// C program to implement one side of FIFO
// This side writes first, then reads
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
    char *myfifo1 = "/tmp/myfifo1";
    char *myfifo2 = "/tmp/myfifo2";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo1, 0666);
    mkfifo(myfifo2, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo1, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2, strlen(arr2) + 1);
        close(fd);

        // Open FIFO for Read only
        fd = open(myfifo2, O_RDONLY);

        // Read from FIFO
        read(fd, arr1, sizeof(arr1));

        // Print the read message
        printf("User2: %s\n", arr1);
        close(fd);
    }
}
```

```
    return 0;
}
```

## **PROGRAM – PIPELINE 2**

```
// C program to implement one side of FIFO
```

```
// This side reads first, then reads
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <fcntl.h>
```

```
#include <sys/stat.h>
```

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
int main()
```

```
{
```

```
    int fd1;
```

```
    // FIFO file path
```

```
    char *myfifo1 = "/tmp/myfifo1";
```

```
    char *myfifo2 = "/tmp/myfifo2";
```

```
    // Creating the named file(FIFO)
```

```
    // mkfifo(<pathname>,<permission>)
```

```
    mkfifo(myfifo1, 0666);
```

```
    mkfifo(myfifo2, 0666);
```

```
    char str1[80], str2[80];
```

```
    while (1)
```

```
{
```

```
    // First open in read only and read
```

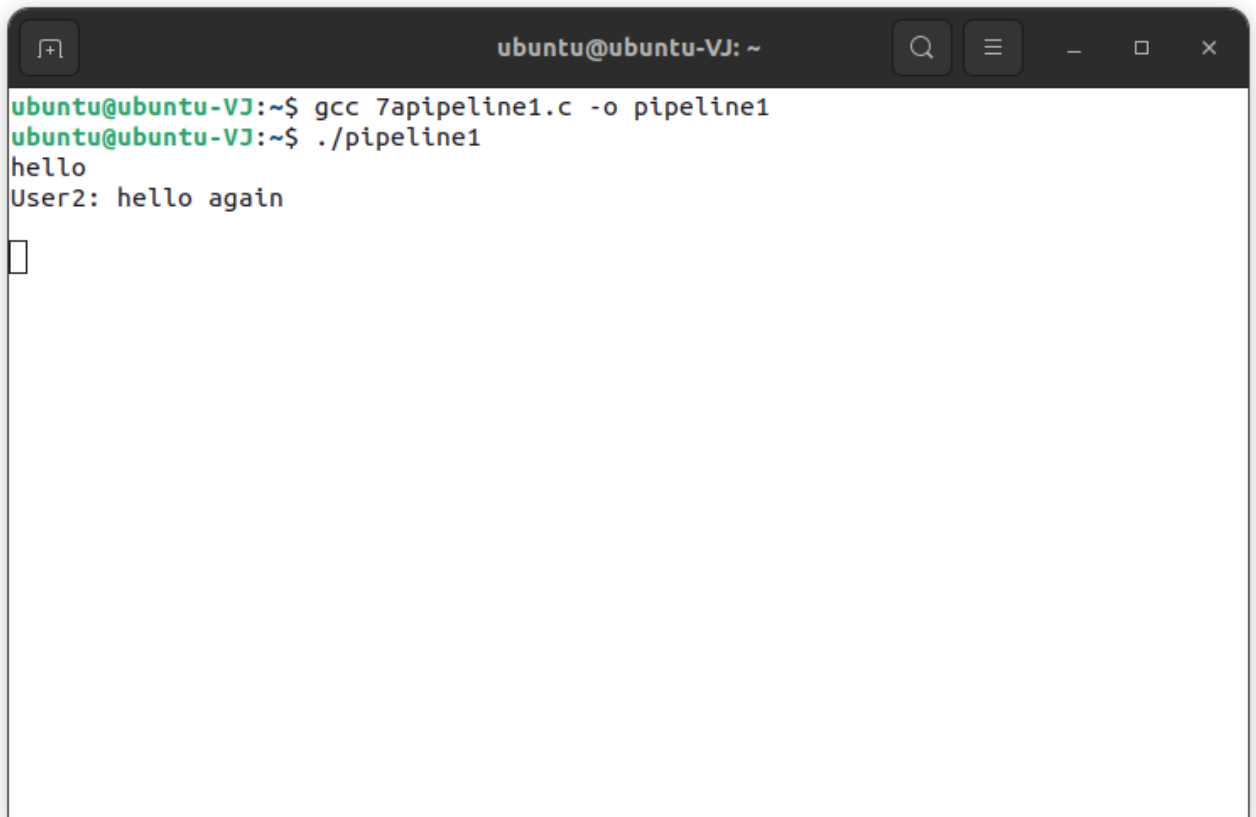
```
    fd1 = open(myfifo1, O_RDONLY);
```

```
    read(fd1, str1, 80);
```

```
// Print the read string and close
printf("User1: %s\n", str1);
close(fd1);

// Now open in write mode and write
// string taken from user.
fd1 = open(myfifo2, O_WRONLY);
fgets(str2, 80, stdin);
write(fd1, str2, strlen(str2) + 1);
close(fd1);
}
return 0;
}
```

**OUTPUT :-**



```
ubuntu@ubuntu-VJ: ~  
ubuntu@ubuntu-VJ:~$ gcc 7apipeline1.c -o pipeline1  
ubuntu@ubuntu-VJ:~$ ./pipeline1  
hello  
User2: hello again  
█
```

A terminal window titled 'ubuntu@ubuntu-VJ: ~' with search, menu, and window control icons. It shows the compilation of '7apipeline1.c' into 'pipeline1' and its execution. The program prints 'hello' and prompts 'User2: hello again'. A cursor is visible on the line following the prompt.



```
ubuntu@ubuntu-VJ: ~  
ubuntu@ubuntu-VJ:~$ gcc 7apipeline2.c -o pipeline2  
ubuntu@ubuntu-VJ:~$ ./pipeline2  
User1: hello  
  
hello again  
█
```

A terminal window titled 'ubuntu@ubuntu-VJ: ~' with search, menu, and window control icons. It shows the compilation of '7apipeline2.c' into 'pipeline2' and its execution. The program prompts 'User1: hello', receives an input, and then prints 'hello again'. A cursor is visible on the line following the output.