NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
ISLAMABAD

OPERATING SYSTEMS LAB SPRING 2019

Lab Manual 05
Inter Process Communication
Named Pipes FIFO

---

1 NAMED PIPES FIFO

A named pipe works much like a regular pipe, but does have some noticeable differences.

1. Named pipes exist as a device special file in the file system.
2. Processes of different ancestry can share data through a named pipe.
3. When all I/O is done by sharing processes, the named pipe remains in the file system for later use.

2 CREATING FIFO

There are several ways of creating a named pipe.

1. It can be done directly from the shell.

```
mkfifo a=rw MYFIFO
```

2. To create a FIFO in C, we can make use of the mkfifo() system call.

FIFO files can be quickly identified in a physical file system by the "**p**" indicator seen here in a long directory listing.
$ ls -l

```
prw-r--r-- 1 root root 23 FEB 14 22:15 MYFIFO|
```

Also notice the vertical bar ("pipe sign") located directly after the file name.

3 USING FIFO THROUGH SHELL

As mentioned before pipe can be created directly from the shell using:

```
mkfifo a
```

Now open 2 different terminals and run the following commands.

```
Terminal 01: ls -l > a
Terminal 02: cat < a
```

where *ls -l* is writing in pipe **a** while *cat* is reading from the pipe and displaying the content.

4 CREATING FIFO IN C++

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include<iostream>
#include<sys/wait.h>
 #include <sys/stat.h>
using namespace std;
int main(){
        int f1, f2;
        //create a pipe with name "pipe_one" and set permissions to 0666
        f1 = mkfifo("pipe_one",0666);
        //check if mkfifo call was successful
        if(f1<0)
                cout<<"Pipe one not created"<<endl;
        else
                cout<<"Pipe one created"<<endl;

        f2 = mkfifo("pipe_two",0666);
        if(f2<0)
                cout<<"Pipe two not created"<<endl;
        else
                cout<<"Pipe two created"<<endl;
        return 0;
}
```

5 OPEN() SYSTEM CALL

An "open" system call or library function should be used to physically open up a channel to the pipe.

```
int open(const char *pathname, int flags);
```

Given a pathname for a file, open() returns a file descriptor.
The argument flags must include one of the following access modes: O_RDONLY, O_WRONLY, or O_RDWR. These request opening the file read- only, write-only, or read/write, respectively.

6 CLOSE() SYSTEM CALL

close() closes a file descriptor, so that it no longer refers to any file and may be reused.

```
int close(int fd)
```

close() returns zero on success. On error, -1 is returned. Not checking the return value of close() is a common but nevertheless serious programming error.

7 COMMUNICATION BETWEEN DIFFERENT ANCESTRY PROCESSES

PROGRAM 01

Write the code in a file and run it in Terminal 01.

```
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <iostream>
using namespace std;
int main(){
        char str[256] = "hello world";
        int fifo_write;
        //open "pipe_one" with WRITE only mode
        // and return its file descriptor
        fifo_write = open ("pipe_one",O_WRONLY);
        //check if open call was successful
        if(fifo_write < 0){
                cout<<"Error opening file"
        }
        else{
                while(strcmp(str, "abort")!=0){
                        cout<<"Enter text: "<<endl;
                        cin>>str;
                        write(fifo_write, str, sizeof(str));
                        cout<<"*"<<str<<"*"<<endl;
                }
                close (fifo_write);
        }
        return 0;
}
```

PROGRAM 02

Write the code in a file and run it in Terminal 02.

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

```
#include <fcntl.h>
#include <iostream>
using namespace std;
int main(){
        char str[256] = "hello world";
        //open "pipe_one" with READ only mode
        // and return its file descriptor
        int fifo_read = open ("pipe_one", O_RDONLY);
        //check if open call was successful
        if(fifo_read < 0){
                cout<<"Error opening file"
        }
        else{
                while(strcmp(str,"abort")!=0){
                        read(fifo_read, str, sizeof(str));
                        cout<<"Text: "<<str<<endl;
                }
                close (fifo_read);
        }
        return 0;
}
```

8 UNLINK()

unlink() deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open, the file is deleted and the space it was using is made available for reuse. If the name was the last link to a file but any processes still have the file open, the file will remain in existence until the last file descriptor referring to it is closed. If the name referred to a socket, FIFO, or device, the name for it is removed but processes which have the object open may continue to use it.

EXAMPLE

```
// creating and unlinking pipes in one file
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <iostream>
#include<sys/wait.h>
 #include <sys/stat.h>
using namespace std;
int main(){
        char str[256] = "hello world";
        int f1, f2;
        f1 = mkfifo("pipe_one",0666);
        if(f1<0)
                cout<<"Pipe one not created"<<endl;
        else
                cout<<"Pipe one created"<<endl;

        f2 = mkfifo("pipe_two",0666);
```

```
        if(f2<0)
                cout<<"Pipe two not created"<<endl;
        else
                cout<<"Pipe two created"<<endl;

        //removing pipes
        unlink("pipe_one");    //remove pipe_one
        unlink("pipe_two");    //removing pipe_two
        return 0;
}
```