Experiment Number: 07

TITLE: Inter-process Communication (IPC) in Linux using FIFO.

OBJECTIVE:

1. To study use of named pipes Linux.
2. To study full duplex inter-process communication in Linux.

THEORY:

Pipes and FIFOs

A pipe is a mechanism for interprocess communication; data written to the pipe by one process can be read by another process. The data is handled in a first-in, first-out (FIFO) order. The pipe has no name; it is created for one use and both ends must be inherited from the single process which created the pipe.

A FIFO special file is similar to a pipe, but instead of being an anonymous, temporary connection, a FIFO has a name or names like any other file. Processes open the FIFO by name in order to communicate through it.

A pipe or FIFO has to be open at both ends simultaneously. If you read from a pipe or FIFO file that doesn’t have any processes writing to it (perhaps because they have all closed the file, or exited), the read returns end-of-file. Writing to a pipe or FIFO that doesn’t have a reading process is treated as an error condition; it generates a SIGPIPE signal, and fails with error code EPIPE if the signal is handled or blocked.

Neither pipes nor FIFO special files allow file positioning. Both reading and writing operations happen sequentially; reading from the beginning of the file and writing at the end.

The Pipe Call

The lower-level pipe( ) function provides a means of passing data between two programs, without the overhead of invoking a shell to interpret the requested command. It also gives us more control over the reading and writing of data.

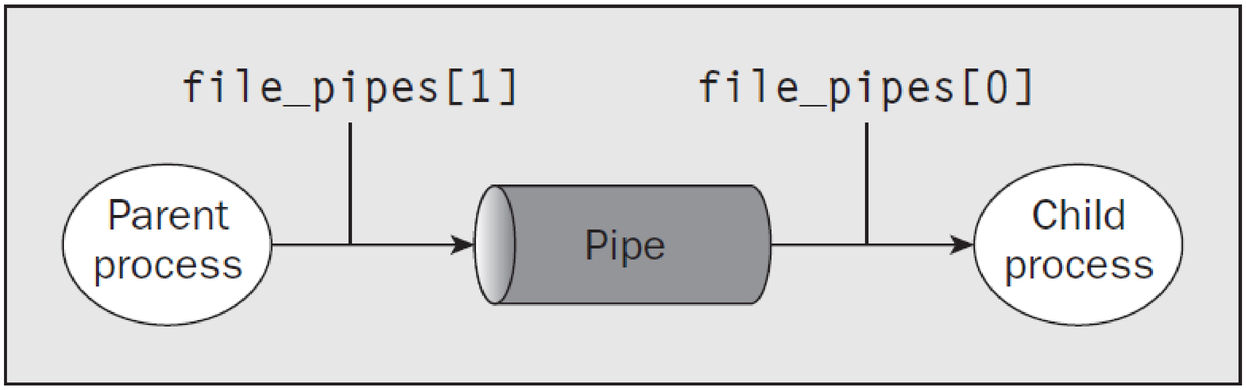
The pipe function has the following prototype:

```
#include <unistd.h>
int pipe(int file_descriptor[2]);
```

It is passed (a pointer to) an array of two integer file descriptors. It fills the array with two new file descriptors and returns a zero. On failure, it returns -1 and sets errno to indicate the reason for failure. Errors defined in the Linux manual page for pipe (in section 2 of the manual) are

- ❑ EMFILE: Too many file descriptors are in use by the process.
- ❑ ENFILE: The system file table is full.
- ❑ EFAULT: The file descriptor is not valid.

The two file descriptors returned are connected in a special way. Any data written to file_descriptor[1]can be read back from file_descriptor[0]. The data is processed in a first in, first out basis. This means that if we write the bytes 1, 2, 3 to file_descriptor[1], reading fromfile_descriptor[0] will produce 1, 2, 3. The illustration is given as below:



Each running program, called a process, has a number of file descriptors associated with it. These are small integers that we can use to access open files or devices. How many of these are available will vary depending on how the system has been configured. When a program starts, it usually has three of these descriptors already opened. These are:

- ❑ 0: Standard input
- ❑ 1: Standard output
- ❑ 2: Standard error

Pipe to a Subprocess

A common use of pipes is to send data to or receive data from a program being run as a subprocess. One way of doing this is by using a combination of pipe (to create the pipe), fork (to create the subprocess), dup2 (to force the subprocess to use the pipe as its standard input or output channel), and exec (to execute the new program). Or, you can use popen and pclose.

The advantage of using popen and pclose is that the interface is much simpler and easier to use. But it doesn't offer as much flexibility as using the low-level functions directly.

Function: **FILE * popen (const char *command, const char *mode)**

The popen function is closely related to the system function; see Running a Command. It executes the shell command command as a subprocess. However, instead of waiting for the command to complete, it creates a pipe to the subprocess and returns a stream that corresponds to that pipe.

If you specify a mode argument of "r", you can read from the stream to retrieve data from the standard output channel of the subprocess. The subprocess inherits its standard input channel from the parent process.

Similarly, if you specify a mode argument of "w", you can write to the stream to send data to the standard input channel of the subprocess. The subprocess inherits its standard output channel from the parent process.

In the event of an error popen returns a null pointer. This might happen if the pipe or stream cannot be created, if the subprocess cannot be forked, or if the program cannot be executed.

Function: **int pclose (FILE *stream)**

The pclose function is used to close a stream created by popen. It waits for the child process to terminate and returns its status value, as for the system function.

FIFO Special Files

A FIFO special file is similar to a pipe, except that it is created in a different way. Instead of being an anonymous communications channel, a FIFO special file is entered into the file system by calling mkfifo.

Once you have created a FIFO special file in this way, any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be open at both ends simultaneously before you can proceed to do any input or output operations on it. Opening a FIFO for reading normally blocks until some other process opens the same FIFO for writing, and vice versa.

The mkfifo function is declared in the header file sys/stat.h.

Function: **int mkfifo (const char *filename, mode_t mode)**

The mkfifo function makes a FIFO special file with name filename. The mode argument is used to set the file’s permissions; see Setting Permissions.

The normal, successful return value from mkfifo is 0. In the case of an error, -1 is returned. In addition to the usual file name errors (see File Name Errors), the following errno error conditions are defined for this function:

EEXIST	The named file already exists.
ENOSPC	The directory or file system cannot be extended.
EROFS	The directory that would contain the file resides on a read-only file system.

ALGORITHM:

PROGRAM (With output and Comments):

CONCLUSION (At least three points):

ASSIGNMENTS:

1. What is difference between pipe and FIFO? Explain at least three points.
2. What are the advantages of FIFO over pipe?
3. Explain the situation where FIFO is appropriate structure used over pipe.
4. State the difference between named and unnamed pipes.

REFERENCES:

1. “Beginning Linux Programming” by Neil Mathew and Richard Stones, Wrox Publications.
2. http://gnu.org/

Tushar B Kute
http://tusharkute.com