Lab06: PIPES

Objectives

1. **Understanding Inter Process Communication**
2. **Using PIPES for IPC**

What Is a Pipe?

We use the term pipe to mean connecting a data flow from one process to another. Generally you attach, or pipe, the output of one process to the input of another.
Most Linux users will already be familiar with the idea of a pipeline, linking shell commands together so that the output of one process is fed straight to the input of another. For shell commands, this is done using the pipe character to join the commands, such as

```
cmd1 | cmd2
```

The shell arranges the standard input and output of the two commands, so that
❑The standard input to cmd1 comes from the terminal keyboard.
❑ The standard output from cmd1 is fed to cmd2 as its standard input.
❑ The standard output from cmd2 is connected to the terminal screen.
What the shell has done, in effect, is reconnect the standard input and output streams so that data flows from the keyboard input through the two commands and is then output to the screen.

The Pipe Call

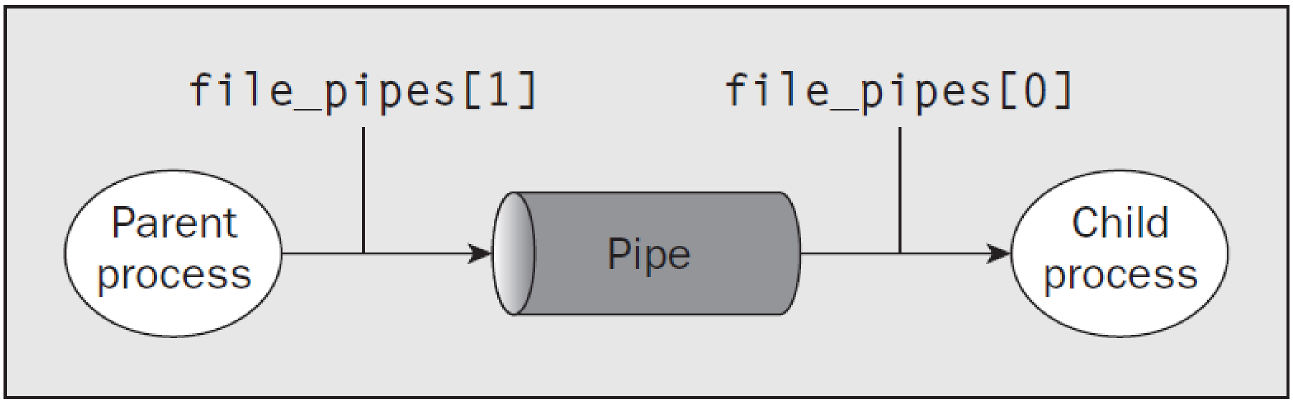
The pipe function has the following prototype:

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

pipe 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 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.

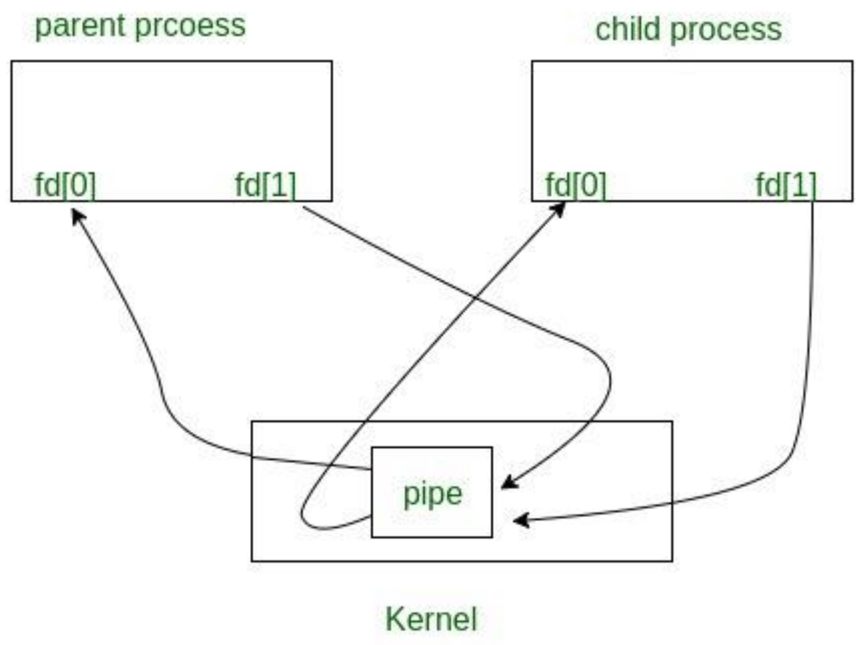
The two file descriptors returned are connected in a special way. Any data written to file_descriptor[1] can be read back from file_descriptor[0]. The data is processed in a first in, first out basis, usually abbreviated to FIFO. This means that if you write the bytes 1, 2, 3 to file_descriptor[1], reading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



Example # 1: Pipe Function

```
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
      char inbuf[MSGSIZE];
      int p[2], i;

      if (pipe(p) < 0)
            exit(1);

      /* continued */
      /* write pipe */

      write(p[1], msg1, MSGSIZE);
      write(p[1], msg2, MSGSIZE);
      write(p[1], msg3, MSGSIZE);

      for (i = 0; i < 3; i++) {
            /* read pipe */
            read(p[0], inbuf, MSGSIZE);
            printf("% s\n", inbuf);
      }
      return 0;
}
```

Example # 2: Pipe Across a Fork



```
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
    char inbuf[MSGSIZE];
    int p[2], pid, nbytes;

    if (pipe(p) < 0)
        exit(1);

    /* continued */
    if ((pid = fork()) > 0) {
        write(p[1], msg1, MSGSIZE);
        write(p[1], msg2, MSGSIZE);
        write(p[1], msg3, MSGSIZE);


       wait(NULL);
    }

    else {
        while ((nbytes = read(p[0], inbuf, MSGSIZE)) > 0)
            printf("% s\n", inbuf);
        printf("Finished reading\n");
    }
    return 0;
}
```