Operating System Lab 05

Assignment 5

Inter-process Communication Using Pipes.

Objectives: To understand what is inter-process communication and implement it using pipes.

Processing steps:

What is inter-process communication?

IPC (inter-process communication) provides a way for the processes to communicate with each other. Processes executing concurrently in the operating system might be either independent processes or cooperating processes. A process is independent if it cannot be affected by the other processes executing in the system.

Pipes

Anonymous pipes (or simply pipes, for short) provide a mechanism for one process to stream data to another. A pipe has two ends associated with a pair of file descriptors - making it a one-to-one messaging or communication mechanism. One end of the pipe is the read-end which is associated with a file-descriptor that can only be read, and the other end is the write-end which is associated with a file descriptor that can only be written. This design means that pipes are essentially half-duplex. Pipes is represented by the symbol **vertical bar,** |.

Anonymous pipes can be setup and used only between processes that share parent-child relationship. Generally the parent process creates a pipe and then forks child processes. Each child process gets access to the pipe created by the parent process via the file descriptors that get duplicated into their address space. This allows the parent to communicate with its children, or the children to communicate with each other using the shared pipe.

Step 1: Understanding Pipe Example

Create a text file named as **student.txt**. Enter the following data in file.

ABC
MNO
XYZ
ABC

Open terminal and run the following commands.

cat student.txt | grep ABC

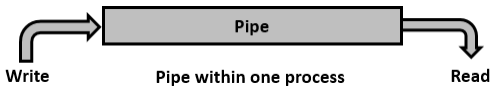
cat command will return the contents of the file and grep command work as filter. It will filter the records based on ABC.

sort student.txt | uniq

sort command will sort the contents of file in ascending order and uniq command will return only unique records.

Step 2: Study pipe() system call

pipe() is a Linux system function. The pipe() system function is used to open file descriptors, which are used to communicate between different Linux processes. In short, the pipe() function is used for inter-process communication in Linux.



The syntax of the pipe() function is:

int pipe(int pipefd[2]);

Parameters: int (Integer) type array **pipefd** consisting of 2 array element to the function pipe(). The pipe() function creates two file descriptors in the pipefd array.

The first element of the **pipefd** array, **pipefd[0]** is used for reading data from the pipe.

The second element of the **pipefd** array, **pipefd[1]** is used for writing data to the pipe.

Return Type: On success, the pipe() function returns 0. If an error occurs during pipe initialization, then the pipe() function returns -1.

Header Files: The pipe() function is defined in the header **unistd.h**.

Step 3: Example 1: Checking the file descriptor values

Run the following program and check the output.

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


int main(void) {
int pipefds[2];


if(pipe(pipefds) == -1) {
perror("pipe");
exit(EXIT_FAILURE);
}
printf("Read File Descriptor Value: %d", pipefds[0]);
```

```
printf("Write File Descriptor Value: %d", pipefds[1]);


    return EXIT_SUCCESS;
}
```

Explaination:

pipfds[2]: Creates two element integer array.
pipe(pipefds[2]): Initialize the file descriptors array and check for errors. The exit() function is used to terminal the program in case the pipe function fails.

Print the value of the read and write pipe file descriptors pipefds[0] and pipefds[1] respectively.

Step 4: How to write to the pipe and read the data from the pipe.
To read from and write data to pipe, read() and write() function is used. The syntax for reading and writing is as follow:

Write: write(pipefds[1], Data to write , size);
Remember: Note that file descriptor 1 is used for writing.

Read: read(pipefds[0], Data to read from buffer, size);
Remember: Note that file descriptor 0 is used for reading.

NOTE: The data read from the pipe will be stored in the **buffer** character array.

Conclusion: At the end of this lab, student will be able to implement inter-process communication using pipe.

Class Activity

1. Write a C program to implement inter-process communication between two processes using pipe.
   Hint: Modify example 1 to write data into pipe and read data from pipe.

Exercise

1. Write a program that implements communication between parent and child process through pipe.
   Implement this in two ways:
   a) Parent process writes data and child process read it.
   b) Child process writes data and parent process read it.

References:

- ”Inter-Process Communication”
  http://www.chandrashekar.info/articles/
  linux-system-programming/introduction-to-linux-ipc-mechanims.html