CPE 460 Laboratory 5: IPC Using Pipes

Department of Computer Engineering
Yarmouk University

Spring 2017

1 Purpose

Understand and use both ordinary and named pipes for inter-process communication.

2 Introduction

Inter-process communication (IPC) is the mechanism whereby one process can communicate with another process, i.e., exchange data. One of the most fundamental IPC mechanisms is the pipe, which symbolizes data flowing sequentially between *related* processes in a pipeline. Pipes are *byte streams* which connect the standard output from one process into the standard input of another process. There are two types of pipes:

- **Ordinary Pipes** are accessible through two file descriptors that are created through the function `pipe(fd[2])`, where `fd[1]` signifies the write file descriptor, and `fd[0]` describes the read file descriptor. Ordinay pipes are only used for *unidirectional* communication between a child and it's parent process, because it dose not have a particular name.

  Blocking Nature in Ordinary Pipes When reading from a pipe: `read()` will return 0 (end of file) when the write end of the pipe is closed. If write end of the pipe is still open and there is no data, `read()` will sleep until input become available. When writing to a pipe: If read end of pipe is closes, a `write()` will fail.

- **Named Pipes** are also called FIFO (first in first out). They have "names" and exist as special files within a file system. They exist until they are removed with `rm` or `unlink()`. Named pipe is referred through it's name only by the reader and writer. All instances of a named pipe share the same pipe name, so they can be used for communication between two unrelated processes.

  Blocking Nature in Named Pipes Named pipes are created using `mkfifo()` system call. Once a named pipe is created, processes can `open()`,`read()` and `write()` to them just like any other file. Using the default system call without `O_NONBLOCK` flag: `read()` will block until a process open the pipe for writing. `write()` will block until a process opens the pipe for reading.

3 Exercise

1 Write and execute the following program:

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

void main(){
    int p[2];
    char buff;
    pipe(p);
    read(p[0],&buff,1);
    printf("Done.\n");
}
```

Why the program blocks and never executes `printf()` statment?

2 Propose two methods to prevent the blocking in the previous part (without remove anything).

3 Write a program that consists of three process *P1*,*P2* and *P3*, where *P1* is the parent of the other two processes. Process *P1* reads five integers from the standad input, and saves them into the **data** element of an instance of the struct below.

```
struct record{
    int data[5];
    int sum;
    double average;
}
```

Then, *P1*writes its instance of the **struct record** into a pipe so that *P2* may read it. *P2* read the struct instance, finds the sum of five integers and save it into **sum** element of the passed struct. Then, *P2* passes the updated struct instance to *P3* via another pipe. Upon receiving the struct instance, *P3* divides the **sum** by 5 and saves the result into **average** element. Again *P3* writes the updated struct instance into a third pipe for *P1* to read. Finally, *P1* reads the updated struct instance from the third pipe, and print its content to the standard output.