CSCI340, Spring 2015
Assignment 3: Benchmarking Signals and Pipes

California State University - Chico
By Bryan Dixon
Adopted from assignment by Fred Kuhns, Washington University in St. Louis[1]

Due Date: Sunday, March 8th, 2015 11:59pm

Introduction

The purpose of this assignment is for you to become more familiar with the concepts of inter-process communication. You'll do this by writing a simple program to benchmark the round trip of sending messages with unix pipes vs signals.

Logistics

The only "hand-in" will be electronic. Any clarifications and revisions to the assignment will be posted on my web page and emailed out to the class.

Hand Out Instructions

I recommend you use an Ubuntu Linux virtual machine to complete this assignment. Alternatively, you can use the jaguar machines or your native Linux install.

Download the file `ipclab-handout.tar` from the course "assignment" page.

Start by copying the file `ipclab-handout.tar` to the protected directory (the *lab directory*) in which you plan to do your work. Then do the following:

- Type the command `tar xvf ipclab-handout.tar` to expand the tarfile.
- Type the command `make` to compile and link some test routines.
- Enter your name in the header comment at the top of `ipc.cc`.

Looking at the `ipc.cc` file, you will see that it contains a skeleton and some variables to get you started. To help you get started, I provided the wrapper for the signal handler from your shell assignment, which is in the `helper_routines.cc` file. You will only need to edit the `ipc.cc` file for this assignment.

General Overview of Pipes

Hopefully you are already familiar with signals at this point from the shell lab. The Unix `pipe()` system call asks the operating system to construct a new anonymous pipe object. This results in two new, opened file descriptors in the process: the read-only end of the pipe, and the write-only end. The pipe ends appear to be normal, anonymous file descriptors, except that they have no ability to seek.[2]

To avoid deadlock and exploit parallelism, the Unix process with one or more new pipes will then, generally, call `fork()` to create new processes. Each process will then close the end(s) of the pipe that it will not be using before producing or consuming any data. Alternatively, a process might create a new thread and use the pipe to communicate between them.[2]

You can have more than one pipe in a process; however, each pipe is its own anonymous file descriptor. Here is a quick example of the creation of a pipe:

```
int fd[2];
pipe(fd);
```

You could now operate on `fd` like a C file descriptor where `fd[0]` would be the input pipe and `fd[1]` would be the output pipe ”file” descriptor. Here’s a quick example of how you could have the child send the parent the message `"Hello World"`[3]:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
        int     fd[2], nbytes;
        pid_t   childpid;
        char    string[] = "Hello, world!\n";
        char    readbuffer[80];

        pipe(fd);

        if((childpid = fork()) == -1)
        {
                perror("fork");
                exit(1);
        }
        if(childpid == 0)
        {
                /* Child process closes up input side of pipe */
                close(fd[0]);

                /* Send "string" through the output side of pipe */
                write(fd[1], string, (strlen(string)+1));
                exit(0);
        }
        else
        {
                /* Parent process closes up output side of pipe */
                close(fd[1]);

                /* Read in a string from the pipe */
                nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
                printf("Received string: %s", readbuffer);
        }

        return(0);
}
```

Your Task

Your task for this assignment is to record how long it takes for a message to come back once a message has been sent from parent to child or child to parent and then output the average, min, max, and elapsed time for all the messages to be passed. You'll also need your parent and child process to identify themselves as the parent or child and display their process id and group id. Here is the expected output for a default run of the program benchmarking pipes:

```
bash$ ./ipc -p
Number of Tests 10000
Child's Results for Pipe IPC mechanisms
Process ID is 4776, Group ID is 1000
Round trip times
Average 0.003187
Maximum 0.126000
Minimum 0.002000
Elapsed Time 32.324000
Parent's Results for Pipe IPC mechanisms
Process ID is 4775, Group ID is 1000
Round trip times
```

```
Average 0.003128
Maximum 0.127000
Minimum 0.002000
Elapsed Time 32.317000
```

Program Specification

- Your program should take the following arguments:
  - -p This argument denotes that your program should benchmark pipes
  - -s This argument denotes that your program should benchmark signals
  - [number]This is a second argument and is the number of tests you want to run. It is optional, and if no value is given then 10000 tests will be run.

Checking Your Work

I have provided some tools to help you check your work.

Reference solution. Just like with the shell assignment, a reference program has been included, `ipcref`, which is a fully functional version of this assignment. *Your program should emit output that is identical to the reference solution*

Makefile. The makefile takes arguments to allow you to test the default number of tests for pipes or signals. To test for pipes type `make pipes` and for signals type `make signals`.

Hints

- Read **Section 3.6.3** of the Operating System Concepts Essentials 2nd edition book.
- Read the references links of this document on pipes.

Evaluation

Your solution will be tested to make sure that it performs the requested behavior of correctly measuring the round trip times for both signals and pipes. Your solution will also be tested to see that it outputs in the same format as the reference program. If your program accomplishes both of these things it will be considered correct.

Hand In Instructions

You only have to change `ipc.cc`. You need to upload `ipc.cc` to the `http://turnin.ecst.csuchico.edu/` page to mark your completion time.

References

[1] Fred Kuhns *CS422: Operating Systems Organization*. Washington University in St. Louis: Spring 2004. `http://www.cs.wustl.edu/~fredk/Courses/cse422/sp04/Projects/Project1.html`.

[2] Wikipedia. "Pipeline (Unix)". Wikipedia, The Free Encyclopedia. 2012. `http://en.wikipedia.org/wiki/Pipeline_%28Unix%29`. Online; accessed 2-February-2014.

[3] The Linux Documentation Project "6.2.2 Creating Pipes in C)". `http://www.tldp.org/LDP/lpg/node11.html`. Online; accessed 2-February-2014.