CS431 Operating Systems
Programming Assignment 2 - Inter-Process Communication Using Signals and Pipes

1 Overview

Modern operating systems typically provide more than one mechanism for inter-process communication. Although you would typically choose one method or the other for a particular application, this exercise will give you practice implementing both signals and ordinary pipes as means of communicating between two processes. You will have two processes, a parent and child, that communicate via a user-defined signal, `SIGUSR1`, sent from the parent that indicates to the child it should retrieve a message the parent sends via pipe. This exercise will be completed in two parts, although you are required to submit only one program.

2 Part 1 - Signals

In this exercise, you will complete the following tasks:

1. Write a program that calls the `fork()` system call to create a child process. The parent process will send messages and the child process will receive them.

2. After spawning a child, the parent process will sleep for 3 seconds (**hint:** consider the `sleep()` library function), then use the `kill()` system call to send a `SIGUSR1` signal to its child (remember, after the call to `fork()`, the parent has the process ID of its child). The parent should write a message to standard output indicating it has sent a signal to its child (and provide the child's process ID).

3. Write and register a signal handler routine that checks to see if the signal received by a process is `SIGUSR1` and sets a (possibly global) variable value accordingly. For the system calls needed to do this part, your program will need to include the preprocessor directive `#include <signal.h>`. The simplest way to register a signal handler is to simply pass the address of the handler function to the system call `signal()` along with the signal to which it is intended to respond. The signal handler function is a function that returns `void` and takes an integer as its only parameter. An example function would look like:

   ```
   void the_handler(int sig_number)
   {
      // your code here
   }
   ```

   To register the signal handler, pass its address to the `signal()` function like this:

```
signal(SIGUSR1, the_handler);
```

Notice there are no parentheses after the function name. You're passing a pointer to the function as an argument to `signal()`, not calling it here. Also, be mindful of the possibility of error conditions resulting from attempting to register a signal handler (check the man page for return values and error numbers), and be sure to handle them as appropriate.

4. Have the child process sleep for 1-second intervals, then check to see if the `SIGUSR1` signal has been received by the signal handler (**hint:** this will happen asynchronously, so your signal handler, once called, should indicate such by setting a variable value to be read later). Once the signal has been received, the child process will write a message to that effect to standard output, then terminate.

3 Part 2 - Pipes

In this exercise, we will use an ordinary pipe to transfer a message between the parent and child processes. Using your code from Part 1, modify the program as follows:

1. Create a pipe by declaring an integer array of length 2 (pipes use a file abstraction, and file descriptors are implemented as type `int`), then passing the array to the `pipe()` system call. As we discussed in class, the first array element (array index 0) is the file descriptor used for reading; the second (array index 1) is used for writing. You will have to use the preprocessor directive `#include <unistd.h>` in order to call `pipe()`. See the textbook pages 141 and 142 (Section 3.7.4.1) for example code on creating and using ordinary pipes.

2. Using the `write()` system call, the parent process will write the message "Hello World!" to the pipe. Make sure the null terminator of the message is written as well by specifying to `write()` that you are writing one character more than the length of the string. The parent will also print a statement to that effect to standard output, including the number of bytes written and the message itself. After writing to the pipe, the parent will next send a `SIGUSR1` signal to its child and `wait()` before terminating. As with writing to the pipe, include an output statement showing the signal has been sent.

3. When the `SIGUSR1` signal is received, the child process will retrieve the message from the pipe by reading it into a buffer, i.e., an array of type `char`, and print to standard output a statement to that effect, including the message that was received and number of bytes read, then terminate.

4 Requirements

Correctness of your program requires the following conditions be satisfied:

- You may build and test your program on any POSIX-compliant platform, e.g., Linux, Solaris, etc. Please let me know what platform you built and tested your code on when you submit.

- Appropriate error checking must be done after attempting to register a signal handler, create a pipe, or fork a new process, with a suitable error message displayed and program termination, if necessary.
- Your program must be commented, to include a header block of comments with your name and assignment information.

5 Deliverables

Although the assignment is described in two parts, both parts make up a single program. Submit the C code source file for your program through the WorldClass assignment dropbox no later than the due date specified in the weekly assignment.

6 Academic Integrity

I recognize that there are only so many ways to write code that meets the requirements of this assignment, and you may consult with other students in this class on the general approach to a solution. Having said that, sharing code, pseudocode, or other implementation details is not permitted, and I expect all work to be your own as evidenced by:

- Program structure, control flow, variable and function names (aside from C library functions, of course), etc. should be manifestly distinct from others' work.
- Your program should be commented in a manner sufficient for me to ascertain that you understand what the code you submit does.