COSC 3360 - Operating Systems Spring 2024

Programming Assignment 1 - Pipes and Process Synchronization

Due Date: Monday, February 26, 2024, 11:59pm CT

Objectives

The goal of this assignment is to help you understand several basic UNIX/Linux primitives, implemented at the process management layer of an OS, such as fork(), pipe(), and execv() as well as simple process synchronization and dataflow systems.

In this assignment, you will implement a program to synchronize, with Unix/Linux pipes, concurrent processes which perform arithmetic operations. You will learn concepts from several computer science disciplines, including data flow architecture, parallel computation, and, of course, operating systems. The input (passed as the first argument of your main program) to your program is a process precedence/dataflow graph specified in the following format:

```
input_var a,b,c,d;
internal_var p0,p1,p2,p3;
  a -> p0;
- b -> p0;
  c -> p1;
+ d -> p1;
  a -> p2;
- d -> p2;
  p0 -> p3;
* p1 -> p3;
+ p2 -> p3;
write(a,b,c,d,p0,p1,p2,p3).
```

`input_var`

declares input variables to be read by your program. Each input value (after it is read) is stored in a process you create.

`internal_var`

declares internal variables whose values will be computed by processes you create. There will be a distinct process to handle the calculation for each internal variable.

To keep the input simple, the values of the input variables to be read by your program will be in the same order they appear in the input_var declaration. There will be no syntax/semantic errors. Also, there are at most 10 input variables and 10 internal variables.

Example for reading input variables:

In the above example,

```
input_var a,b,c,d;
```

the values of a,b,c,d are stored in an input file or stdin. For example, an input file may contain:

```
2,3,1,8
```

or

```
2 3 1 8
```

Your program should work for different sets of values for the input variables.

Precedence/Dataflow Graph

A variable name beginning with ‘p’ represents an internal variable. Other variable names represent input variables. A pipe connects an input variable i (also stored in a process) or process px to another process py if process py depends on i or px, that is, a value is sent from i or px to py. For example,

```
  a -> p0;
```

means that the value of input variable a is sent via a pipe to p0.

```
- b -> p0;
```

means that the value of input variable b is sent via another pipe to p0, which then performs the arithmetic operation a-b. Note that for this arithmetic operation to start, both a and b are available in the respective pipes for reading by process p0.

Another example:

```
  p0 -> p3;
* p1 -> p3;
+ p2 -> p3;
```

means that the values in processes p0, p1, and p2 are passed to process p3 via three different pipes, and then process p3 performs the arithmetic operation p0*p1+p2. Note that for this arithmetic operation to start, all the values computed by the three processes p0, p1, and p2 must be available in the respective pipes for reading by process p3.

Therefore, the above process precedence/dataflow graph corresponds to the operations in the following program:

```
input_var a,b,c,d;
process_var p0,p1,p2,p3;
     p0 = a - b;
     p1 = c + d;
     p2 = a - d;
     p3 = p0 * p1 + p2;
     write(a,b,c,d,p0,p1,p2,p3).
```

Note that there are no parentheses in the arithmetic operations. Also, arithmetic operations are performed from left to right. The specification ends with a ‘write’ statement.

The output of your program consists of a printout of the values of all input and internal variables after all the arithmetic operations have completed.

Submitting the program:

Submit your code and associated documents via Canvas.

Notes

Before you start your assignment, familiarize yourself with the way C/C++ programs handle arguments that are passed to them by execv() and with the UNIX functions fork(), pipe() and dup().

The programs you turn in should be well documented. Each C/C++ function should start by a brief description of its purpose and its parameters. Each variable declaration should contain a brief description of the variable(s) being declared.

The grading rubric is 70% for correctness, 20% for efficiency, and 10% for documentation/comments.

Late penalty: -5 points (maximum number of points: 100 points) per day after the submission deadline. Note that you have 3 grace days to use (wisely) for the entire semester. For example, if you use one grace day for this assignment, then you can submit it one day late with no penalty (indicate the use of one grace day on page 1 of your submission).

This is an individual assignment. Code-level plagiarism or collaboration is forbidden. If you copy code(s) from any source, including those from students who previously took the course and any online source, please explicitly cite it on page 1 of your submission. Even with explicit citations, the logical similarity to any codes available online or submitted either this year or previous years should be less than 75%. Otherwise, you may end up receiving as a minimum penalty a '0' for this assignment and a maximum penalty of an 'F' for the course plus additional disciplinary actions.