COSC 3360/6310 - Operating Systems Fall 2020

Programming Assignment 1 - Pipes and Process Synchronization

Due Date: Wednesday, October 5, 2020, 11:59pm CDT

Objectives

This assignment will help you understand several basic UNIX/Linux primitives, implemented at the process management layer of an operating system, 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, computer arithmetic, 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;
vertex v0 = PLUS, v1 = MINUS, v2 = DIVIDE, v3 = TIMES;
  a  -> v0;
  b  -> v1 -> v0;
  c  -> v0;
  c  -> v1;
  v1 -> v0;
  c  -> v2;
  d  -> v2 -> v3;
  a  -> v3;
  v0 -> v3;
write(a,b,c,d,v0,v1,v2,v3).
```

`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.

`vertex`

declares vertices representing arithmetic operators (PLUS, MINUS, TIMES, DIVIDE) in the precedence graph whose values will be computed by processes you create. There will be a distinct process to handle the calculation for each vertex 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. However, there may be duplicates, that is, an edge is specified more than once but only a single pipe should be created for these duplicates. Also, there are at most 10 input variables and 10 vertices.

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:

```
1 2 3 4
```

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

Precedence/Dataflow Graph

A variable name beginning with 'v' represents a vertex or vertex variable. Other variable names represent input variables. In the example above,

```
vertex v0 = PLUS, v1 = MINUS, v2 = DIVIDE, v3 = TIMES;
```

v0 performs the + operation, v1 performs the - operation, v2 performs the / operation, and v3 performs the * operation. Note that the MINUS and DIVIDE operators are binary (each takes two operands or values) so each operation needs two input values; while the PLUS and TIMES operators may each have two or more inputs. For the MINUS and DIVIDE operators, the input with the larger value is the first operand. If the input values are equal, then either value can be the first operand. Note that for MINUS, the output is always 0; for DIVIDE, the output is always 1. For example, if two numbers, 9 and 3, are inputs for the DIVIDE operator, then the arithmetic operation is 9/3 = 3. Similary, for the MINUS operator, the arithmetic operation is 9 - 3 = 6.

A pipe connects an input variable i (also stored in a process) or process vx to another process vy if process vy depends on i or vx, that is, a value is sent from i or vx to vy. For example,

```
a -> v0;
```

means that the value of input variable a is sent via a pipe to v0. Note that for an arithmetic operation in a vertex vx to start, all the inputs to this vertex must be available, that is, these inputs are in the respective pipes for reading by process vx.

In the above example snippet:

```
a  -> v0;
b  -> v1 -> v0;
c  -> v0;
c  -> v1;
v1 -> v0;
```

means that the value in process a (with an input value) is passed to process v0 via a pipe; the value in process b (with another input value) is passed to process v1 via another pipe and then the value calculated in v1 (after performing the MINUS operation c - b = 3 - 2 = 1) is passed to process v0 via yet another pipe; and the value in process c (with yet another input value) is passed to processes v0 and v1 via two other pipes. Note that the last line in this example snippet specifies the same edge (duplicate) from v1 to v0 in the second line, so no new pipe is created here between v1 and v0. Then process v0 performs the PLUS operation on the values from a, c, and v1 (1 + 3 + 1).

Therefore, the above process precedence/dataflow graph corresponds to the following arithmetic formula:

```
a * (a + c + (c - b)) * (d / c)
```

Note that no parentheses need to be specified by the precedence graph with the arithmetic operations. Also, arithmetic operations are performed from left to right; hence you should read the specifications of the precedence graph in the order they appear. The specifications end with a 'write' statement.

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

Submitting the program:

For submission guidelines, please visit Blackboard/TAs' instructions.

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.

Code-level plagiarism 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. 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 a 0 for assignment 1.