COSC 3360/6310 - Operating Systems Fall 2019

Programming Assignment 1 - Pipes and Process Synchronization

Due Date: Monday, October 2, 2019, 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 logic operations. You will learn concepts from several computer science disciplines, including data flow architecture, parallel computation, Boolean logic, 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 = OR, v1 = NOT, v2 = AND, v3 = IMPLY;
  a  -> v0;
  b  -> v1;
  v1 -> v0;
  c  -> v2;
  d  -> v2;
  v0 -> v3;
  v2 -> 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 logic operators (NOT, AND, OR, IMPLY) 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. 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:

```
T F T F
```

where T means true and F means false. 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 = OR, v1 = NOT, v2 = AND, v3 = IMPLY;
```

v0 performs the OR operation, v1 performs the NOT operation, v2 performs the AND operation, and v3 performs the IMPLY operation. Note that NOT is unary so there can only be one input; IMPLY is binary so it takes two inputs; while OR and AND may each have two or more inputs.

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 a logic 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:

```
  a  -> v0;
  b  -> 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 the value in process v1 (after performing the NOT operation on b) is passed to process v0 via yet another pipe. Then process v0 performs the OR operation on the values from a and v1.

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

```
(a OR (NOT b)) IMPLY (c AND d)
```

which is equivalent to:

```
NOT (a OR (NOT b)) OR (c AND d)
```

Note that no parentheses need to be specified by the precedence graph with the logic operations. Also, logic 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 logic operations have completed.

Submitting the program:

For submission guidelines, please visit the TA/course webpage.

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.