COSC 3360/6310 - Operating Systems Fall 2015

Programming Assignment 1 - Process Synchronization with Unix/Linux Pipes

Due Date: Friday, October 9, 2015, 11:59pm

In this assignment, you will implement a program to synchronize concurrent processes which perform arithmetic operations. This is especially applicable for execution in dual or quad-core computer systems.

You will learn concepts from several computer science disciplines, including compilers, data flow architecture, parallel computation, and, of course, operating systems. The input to your program is illustrated by the following example:

```
main()
{  input_var a,b,c,d;
   internal_var p0,p1,p2,p3;

   read(a,b,c,d);
   cobegin
      p0 = a - b;
      p1 = c + d;
      p2 = a - d;
   coend;
   p3 = (p0 + p1) * p2;
   write(a,b,c,d,p0,p1,p2,p3);
}
```

The syntax of the concurrent program follows that of C with the following extensions/modifications.

`input_var`

declares input variables to be read by your program using "read". Each input value is stored in a process you create.

`internal_var`

declares internal variables whose values will be computed by a process you create. There will be a distinct process to handle the calculation for each internal variable. Statements within a cobegin-coend pair can be executed concurrently. Therefore, processes which compute the internal variables in these concurrent statements can be executed concurrently.

In the above example, processes to compute the values of p0, p1, and p2 can be executed concurrently. However, these 3 processes must complete before the process for computing p3 can start. There can be several levels of nesting cobegin-coend pairs and your program should parse these statements correctly. There may be one level of parentheses in one arithmetic operation, and this also imposes precedence constraints on the processes' arithmetic operations.

You may want to create an internal precedence graph to store the dependencies among the processes. To synchronize these concurrent processes, you are to use Unix/Linux pipes. For the

above example code, there is a pipe (for sending the input value of a) from process a to process p0, and another pipe (for sending the input value of b) from process b to process p0. Process p0 performs the arithmetic operation

"a - b" and sends the result via another pipe to process p3 for it to perform the arithmetic operation "(p0 + p1) * p2".

The output of your program consists of a printout of the values of all input and internal variables as specified by "write".

Notes:

To keep the input simple: The variables in read(...) will be in the same order they appear in the input_var declaration. There will be no syntax/semantic errors.

There are at most 10 internal variables.

Example for reading input variables:

```
read(a1,a2,a3,a4) /* values of a1..a4 are stored in an input file or stdin */
```

The input file contains:

```
2,4,1,9
```

or

```
2 4 1 9
```

Also, no constants (5 in this example) are allowed in right-hand-side expressions like p1 = a1 + 5. Only input variables or internal variables are allowed.

Submitting the program:

For submission guidelines, please visit the TAs' webpage. Also, check the forum there for common questions and answers.