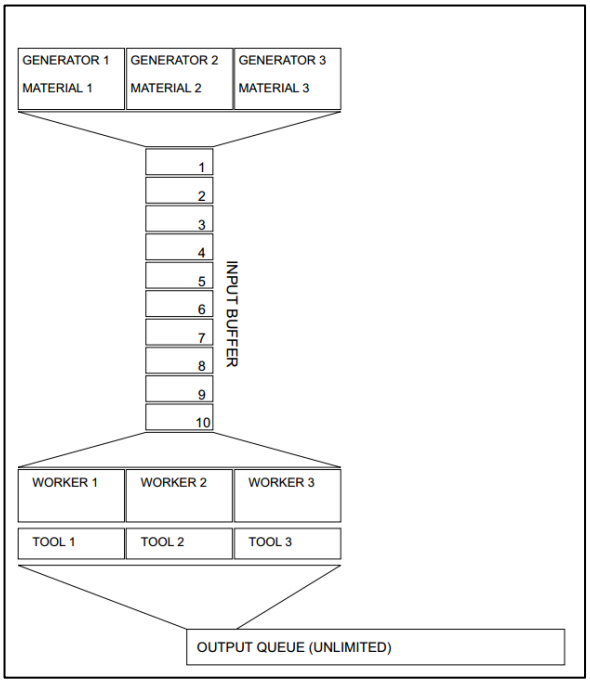
Assignment 2: An Extended Case of the Producer-Consumer Problem

Due date: Tuesday, November 4th, 2014 23:59:59

Value: 10%

Problem Description



- There are 3 generators, and each produces a unique kind of material independently.
- All these materials are stored in an input buffer with size 10 before they are forwarded to the operators.
- We have 3 operators with same priority who are responsible for producing the products based on these materials.
- Each product needs 2 different kinds of materials.
- Each time an operator needs 2 tools for this purpose.
- There are 3 tools in total provided for these operators.
- An operator can only process one product at one time.
- When an operator gets both the materials and tools, he can produce a product within a limited time varied from 0.01 second to 1 second. Otherwise, he has to wait until all the necessities are met.
- He can grab the materials or tools first, it does not matter, but he can only get one thing at one time.
- If an operator decides to make another product before he starts to make the current product , he can put the materials and tools back and re-get the new materials and tools.
- The operator has to put the tools back after he finishes a product because other operators may need these tools.
- All the products are put into a size-unlimited output queue.
- An operator cannot start a new product before he puts the product into the output queue.
- Some restrictions may apply to the output queue:
- No same products can be next to each other in this queue. We say that two products are same if they are made from the same kinds of materials.
- The difference of the number of any two kinds of products produced should be less than 10, for example, we can have 10 of product A and 15 of product B, but it is not allowed if we have 10 of A and 21 of B because the difference is 11 which is larger than 10.

Problem Requirements

You will write a program to simulate this model using **C/C++ on a UNIX system** using a multi-threaded technique. Your program should not stall in a dead cycle at any time. During the execution of your program, you should provide the following information dynamically:

1. For each material, how many are generated?
2. The status of the input buffer
3. For each kind of product, how many are produced?
4. The status of the output queue.
5. How many times deadlock happens?

You also need to provide the following functions:

1. Pause and resume the program at any time.
2. Make the number of operators and tools adjustable.

Please feel free to provide additional information and functions that you consider useful.

Hints:

- **Analyze the problem thoroughly before you begin programming**. The success of your solution will depend on a smart design.
  - Identify shared resources and possible race conditions that may lead to **deadlock**
  - **Keep in mind that some deadlock conditions are not explicitly mentioned in the problem description but may nonetheless occur, or may result from your design decisions. For example, what would happen if the input buffer contains too many of the same kind of material?**
  - Think about how you can prevent deadlock from happening using semaphores and mutex variables
  - Completing this step will aide you in the preparation of your design document and will help you to obtain a good solution.

What to submit

You will submit the following on **courses.cs.sfu.ca :**

1. **Report.pdf –** A document (2 pages) that describes the internal design of your program, focusing on your implementation of threads, mutex variables and semaphores to prevent possible deadlock. It may be helpful to structure your report according to the sections of your code. Make sure to explain any assumptions made in the design of your solution.

2. **HW2.zip (or .rar, or .tar) –** A single archive containing **all your source code plus a Makefile,** in a directory called `hw2`.

   Your executable program will be called `hw2` and running `make` produces the executable. Running `./hw2` will start your program. If necessary, submit a README file that contains any special instructions or information you wish to provide for the grader.

Grading criteria

Your homework assignment is worth 10 points, and will be marked according to the following general criteria

It is your responsibility to ensure the **program compiles and runs smoothly in Linux, with the kind of environment available at CSIL**. Your program must compile and run or you will get a mark of zero on your assignment.

Correctness of your solution (8 points)

The most important thing is to make sure your program works correctly. Do not worry about your programming style or efficiency. But it does matter if you provide a smarter internal design (approach) and user-friendly interface.

A correct solution:

- Identifies and resolves race conditions and deadlock, and the program never stalls in deadlock **(4 marks)**
- Correctly models the **independent** behavior of the generators and workers as described in the problem description, and makes reasonable design assumptions **(3 marks)**.
- Provides a user-friendly interface **(1 mark).**

Note: Although programming style will not be explicitly marked in this assignment, please make some effort to ensure that your code is organized, indented and commented to an extent that it is easy to read and understand.

Design document (2 points)

Your document (2 pages long) must describe the internal design of your program:

- **Generator:** A detailed description of your design and implementation of the independent behavior of **generators,** their access to shared resources, and explain any design assumptions made. Identify all possible deadlock conditions and how you used mutex/semaphores to resolve them. How did you model the pause behavior?
- **Operator:** same as above

Late submissions will be penalized 10%, and will not be accepted after more than two days late.

Assignment grades will be open for review for ***one week only.***

Finally, a reminder that plagiarism will result in an automatic failing grade for the course.