**Laboratory 3  
Due: 29 September 2016 (through Blackboard)**

This laboratory will result in your ability to build a simple program that uses semaphores and multiple threads. I suggest that you use the program(s) from Laboratory 2 as a starting point, and reference the semaphore example code posted with the lecture material.

## Steps:

1. Create two new projects named “Lab3-prog1”, and “Lab3-prog2” inside the GIT repository for your laboratory work.
2. Create a new program, *prog1* inside the Lab3-prog1 project.
   1. Modify the program(s) from Laboratory 2 to:
      1. Prompt for the number of child threads (don’t create any child processes)
      2. Create that number of child threads (print out a message identifying which thread has been created)
      3. **In addition** to waiting for the SIGUSR1 interrupt as in Laboratory 2, the child threads should all wait on the same semaphore.
      4. Once they unblock from the **sem\_wait** function, each thread should print a message identifying which thread has unblocked and then sleep for five seconds.
      5. They should then loop back and wait for the semaphore again (in an infinite loop).
      6. You can kill the process by using the “kill –s SIGUSR1 *process\_id*” command from another window.
3. Create a second program, *prog2* inside the Lab3-prog2 project. This program will “wake up” the blocked threads from the first program.
   1. The second program should
      1. Print its pid.
      2. Read from the command prompt the number of threads of *prog1* to wake. Prompt “How many threads do you want to wake up (enter 0 to exit)?”
      3. Your main thread will then **increment** the semaphore object that *prog1* used by the number received.
      4. Loop back to the prompt so that you can keep waking threads. Exit when 0 is entered.
4. Test your programs.
   1. When *prog1* is running, the threads will wait on the semaphore to increment.
   2. With *prog1* already running, run *prog2.* As soon as you enter the number of threads to wake, *prog1* should wake them. You should see the wakeup messages from each child thread.
5. Save and commit the changes to your repository.
6. Go to the working directory for your GIT repository (the one named with your Algonquin username) and zip it.
   1. If you open the “Window>Open Perspective>Other…>Git Repository Exploring”

You can see where it is physically stored.

1. Submit your laboratory for grading by submitting it online using Blackboard under the “Handing in Work” link, with the zip file as an attachment.

## Grading Scheme

Marked out of 30. 15 points for each of the programs.

Each program marked with the following rubric.

|  |  |
| --- | --- |
| **Criteria** | **Mark** |
| Superior capability. Submission meets or exceeds expected standards | 15 |
| Satisfactory capability, acceptable product/result | 10 |
| Marginal capability, substandard product/result | 5 |
| No capability, unacceptable product/result. Work not submitted | 0 |