Lab 4 – Threads

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 Lab 3 as a starting point and reference the semaphore example code posted with the lecture material.

Steps:

1. Create a new Momentics workspace named: **cst8244_lab04**
2. Create two new projects named **thread-factory**, and **thread-waker**. Create both projects as: QNX Projects > QNX Executable
3. Create a local git repository; add both projects to git and make your initial commit.
4. Create a new program (or rename existing C file), **thread_factory.c** inside the thread-factory project.
   a. Modify the program(s) from Lab 3 to:
      i. Prompt for the number of child **threads** (don't create any child processes)
      ii. Create that number of child threads (print out a message identifying which thread has been created)
      iii. **In addition** to waiting for the SIGUSR1 interrupt as in Lab 3, the child threads should all wait on the same semaphore.
      iv. 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.
      v. They should then loop back and wait for the semaphore again (in an infinite loop).
      vi. You can kill the process by using the "kill –s SIGUSR1 *process_id*" command from another window.
5. Create a second program, **thread_waker.c** inside the thread-waker project. This program will "wake up" the blocked threads from the first program, thread_factory.
   a. The second program should
      i. Print its pid.
      ii. Read from the command prompt the number of threads of *thread_factory* to wake. Prompt "How many threads do you want to wake up (enter 0 to exit)?"
      iii. Your main thread will then **increment** the semaphore object that *thread_factory* used by the number received.
      iv. Loop back to the prompt so that you can keep waking threads. Exit with success (EXIT_SUCCESS) when 0 is entered.
6. Test your programs.
   a. When *thread_factory* is running, the threads will wait on the semaphore to increment.
   b. With *thread_factory* already running, run *thread_waker.* As soon as you enter the number of threads to wake, *thread_factory* should wake them. You should see the wakeup messages from each child thread.

Deliverables

Before making the zip-file deliverable (next item), please action:

- Format your source code to make it easier for me to read. Momentics IDE will do this for you: Source -> Format
- Verify your projects build cleanly. Please action: a) Project -> Clean… and b) Project -> Build All

Prepare a zip-file archive that contains the following items:

1. A “README.txt” file reporting the status of your lab. Follow this template:

   Title {give your work a title}

   Status
   {Tell me that status of your project. Does your program meet all of the requirements of the specification? Does your program run, more importantly, does your program behave as expected? Does your program terminate unexpectantly due to a run-time error? Any missing requirements? A small paragraph is sufficient.}

   Known Issues
   {Tell me of any known issues that you’ve encountered.}

   Expected Grade
   {Tell me your expected grade.}

2. Export your Momentics workspace as a zip-archive file.
   Momentics IDE provides a wizard to export your projects:
   File > Export… > General > Archive File > Next > Select All > Click “Browse…” > Save As: **cst8244_lab4_*yourAlgonquinCollegeUsername*.zip** > Save > Finish
3. Make a video screen capture (i.e. movie) showing the run-time behaviour of *thread-factory* and *thread-waker*. Capture this scenario:
   i. Run *thread-factory* and create ten threads when prompted.
   ii. Run *thread-waker* and wake ten threads when prompted.
   iii. In the *thread-waker* Console window, wake five threads when prompted.
   iv. In the *thread-waker* Console window, wake six threads when prompted.
   v. In the *thread-waker* Console window, enter 0 to exit. In the Neutrino terminal window, run **pidin** and verify that *thread-waker* is no longer a running process.
   vi. From the Neutrino terminal, run **kill -s USR1 *pid-of-thread-factory***. This command terminates the *thread-factory* process.
   vii. Run **pidin** from the Neutrino terminal to verify that *thread-factory* is no longer a running process.
   viii. Show the Console window for *thread-factory*.

To produce your movie, select an appropriate tool for your environment. Mac users, like me, can use Quicktime. I would welcome suggestions from the Windows PC users and Linux folks too.

Upload and submit your zip-file to Brightspace before the due date.

Reference Screencast

Your solution's run-time behaviour is to match the run-time behaviour as seen in the *Reference Screencast for Lab #4*, posted on Brightspace in the *Lab #4* module.