# Mini Project 2: Threads and Syncronization

**Due: March 4th, 2020**

---

## The Assignment

Like in the previous mini project, you may work in groups to complete this lab.

1. **Request Scheduling using Threads and Synchronization**
   In this assignment, we will use threads and synchronization to implement an useful application based on the Producer Consumer problem studied in class. This lab has two parts, one needs to be completed in Java and another in C/C++ with the overall goal of exposing you to threading and synchronization concepts in two widely used programming languages.

   The goal of the lab is to implement a multi-threaded request scheduler that is similar to how the popular Apache web server schedules request. You do not need to have a knowledge of Apache to complete the lab - simply use the problem specification below.

   Assume that request scheduling involves a Master thread and N Slave threads. The master thread "listens" for requests and inserts it into a request queue. Slave threads wait for requests to arrive into the request queue and upon the arrival of a new request, any idle slave thread that is waiting for a request may take the request and process it.

   Your goal is to implement this technique for dispatching requests using a bounded buffer producer-consumer framework. Assume that the request queue is a circular buffer of size N (i.e., an array of size N). The method involves a single producer, which is the master thread, and N consumers, which are the slave threads.

   The master thread will sleep for a random short duration and produce a request. Each request has a sequentially increasing request ID and a randomly chosen request length (assign each new request a random length between 1 and M seconds). The master thread then inserts the request into the queue and goes back to sleep for a random short duration before it produces another request. Of course, if the request queue, which is a bounded buffer, is full, the master thread must wait before it can insert the request into the queue.

Each slave thread can be idle or busy. When a slave thread is idle, it acts as a consumer waiting for a new request in the request queue. After it consumes a request from the queue, the slave thread will be busy for a duration that is equal to the request length for that request. The busy state of a slave thread should be emulated by making the thread sleep for that duration. Upon completing the request, the slave thread goes back to idle thread and attempts to consume a pending request from the request queue. If the queue is empty, the slave thread must wait, like a consumer in the producer consumer problem.

In part 1 of this lab, you will implement the above problem using Java and Monitors. Implement the bounded buffer producers and consumers using concepts discussed in class. Run your program with a single producer (Master thread) and N consumers (slave threads). N should be a configurable parameter that you specify as input to your program. You can also specify other inputs such as M, the max duration of a request and parameter that indicate how long a producer should sleep before producing the next request. The master and slave threads should generate and consume requests as discussed above. Have the master and slave threads print useful messages that indicate their actions. For example, the producer should print messages such as
Producer: produced request ID n, length t seconds at time CURRENT-TIME
Producer: sleeping for X seconds

The consumer should print messages such as
Consumer i: assigned request ID n, processing request for the next t seconds, current time is CURRENT-TIME
Consumer i: completed request ID n at time CURRENT-TIME

The second part of the lab involves solving the same problem in C++ (use of C is allowed if you prefer it over C++) using the pthreads library. Since C and C++ do not support monitors, you should use semaphores to implement your solution. Like before, your code should implement a single producer (master) thread and N consumer (slave) threads. The master and slave threads should print instructive messages like in part 1 to indicate their current actions.

**Helpful references** IF you are not familiar with pthreads, be sure to review this reference on Pthreds programming. A brief tutorial is also available.

There are many tutorials on Java threads and syncronization such as here. Oracle provides a concurrency tutorial here

# How to Turn in Lab 2

**All of the following files must be submitted on Canvas as a zip file to get full credit for this assignment.**

1. Your zip file should contain a copy of all source files. Please include the source code for parts 1 and 2 in separate directories in the zip file
2. Your zip file should contain a copy of a README file identifying your lab partner (if you have one) and containing an outline of what you did for the assignment. It should also explain and motivate your design choices. Explain the design of your program and how syncronization works. Keep it short and to the point.

   If your implementation does not work, you should also document the problems in the README, preferably with your explanation of why it does not work and how you would solve it if you had more time. **Of course, you should also comment your code. We can't give you credit for something we don't understand!**

3. Your README in the zip file should contain build instructions stating exactly how to compile your code on the Linux for both the Java and C/C++ implementations. Preferably, these will be in the form of Makefiles.
4. Your README file should contain run instructions stating exactly how to execute your compiled code on the Linux. for both the Java and C/C++ implementations. These instructions should include a description of any command line arguments your program expects or accepts. If your program prints a usage message, then it is sufficient to say how to get your program to print that message. If max request duration or producer sleep time can only be adjusted by editing your source code, those values should be global variables and you must state their locations in your code.
5. Add a brief discussion describing your experience implementing this problem in Java and C/C++. Comment on the amount of effort and ease of coding the problem in different languages.
6. Finally, your zip file should contain a copy showing sample output from your programs.
7. Individual Group Assessment

   A percent of your lab grade will come from your participation in this project as a member of your group.

   *What you need to turn in* (each person individually):

Include in your zip file a copy of your assessment of the division of labor in the group in the format shown below. For a 2 person group, if you give yourself a 50% and your partner gives you a 50%, you will get the full credit for group participation. If you give your partner a 40% and your partner gives himself or herself a 40%, he or she will get fewer points for group participation. And so on...

8. **Note:** We will *strictly* enforce policies on cheating. Remember that we routinely run similarity checking programs on your solutions to detect cheating. Please make sure you turn in your own work.

   You should be very careful about using code snippets you find on the Internet. In general your code should be your own. It is OK to read tutorials on the web and use these concepts in your assignment. Blind use of code from web is strictly **disallowed**. Feel free to check with us if you have questions on this policy. And be sure to document any Internet sources/ tutorials you have used to complete the assignment in your README file.

9. **Late Policy:** Please refer to the course syllabus for late policy on labs assignments. This late policy will be strictly enforced. Please start early so that you can submit the assignment on time.