# EE/CSCI 451
# Spring 2016
# Programming Homework 3

**Assigned: February 8, 2016**
**Due: February 19, 2016, before 11:59 pm, submit via blackboard**
**Total Points: 50**

## 1 Producer-consumer [25 points]

The producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-thread synchronization problem. In this assignment, you will simulate the producer-consumer problem using mutex and condition variable. Producer makes cookie and puts cookie on the shelf. Consumer buys cookie and takes cookie away from the shelf.

- There are **two** consumers and one producer. Each producer or consumer is implemented as a thread.
- The capacity of the shelf is 10.
- When the producer checks the number of cookies on the shelf, (1) if the number is less than 9, he puts 2 cookies on the shelf; (2) if the number is 9, he puts 1 cookie on the shelf.
- When the shelf is not empty, a consumer thread can only take 1 cookie at a time.
- If the producer thread has put $\geq 30$ cookies on the shelf, it can be terminated.
- If a consumer thread has taken 15 cookies away from the shelf, the consumer thread can be terminated.
- Whenever the number of cookies on the shelf changes, you need print a message in the following format.
  - Producer has put 2 cookies; # of cookies on the shelf changes from 0 to 2.
  - Consumer 2 has taken 1 cookies; # of cookies on the shelf changes from 2 to 1.
  - Producer has put 4 cookies; # of cookies on the shelf changes from 1 to 3.
  - Consumer 1 has taken 1 cookies; # of cookies on the shelf changes from 3 to 2.
  - Consumer 1 has taken 2 cookies; # of cookies on the shelf changes from 2 to 1.
  - Consumer 1 has taken 3 cookies; # of cookies on the shelf changes from 1 to 0.
  - ...

1. **Using mutex only [10 points]**
   In this version, when each thread intends to check/update the number of cookies on the shelf, the thread needs to acquire the mutex first in order to avoid any race condition. Here, only mutex is used. Name this program as p1a.c.
2. **Using mutex and condition variable [15 points]**
   In this version, when the producer checks the number of cookies on the shelf, if the number is 10, producer goes to sleep; if the number is 0, producer puts 2 cookies on the shelf and sends a broadcast signal to wake up the consumers. When a consumer checks the number of cookies on the shelf, if the number is 0, the consumer goes to sleep. Initially, all the threads are awake. Name this program as p1b.c.

# 2 Parallel K-Means [25 points]

In PHW 2, you implemented the $K$-Means algorithm using Pthreads. In this problem, you will use mutex and condition variable to realize synchronization instead of iteratively joining the threads. Let $p$ denote the number of threads that you need. In this version, you only create and join $p$ threads once (not iteratively create and join the threads). This version of $K$-Means algorithm has the following steps:

1. Initialize a mean value for each cluster.
2. Partition and distribute the data elements among the threads. Each thread is responsible for a subset of data elements.
3. Each thread assigns its data elements to the corresponding cluster. Each thread also locally keeps track of the number of data element assigned to each cluster in current iteration and the corresponding sum value (these are local intermediate data for each thread which can be read by other threads but can not be updated by other threads).
4. Let us use $r$ to denote the number of threads which have completed the work for the current iteration. At the beginning of each iteration, $r = 0$. When a thread finishes its work for the current iteration, it checks the value of $r$. (Note that $r$ is a shared variable, a mutex is needed whenever any thread tries to read/write it.)
   - If $r < p - 1$, $r \leftarrow r + 1$, the thread goes to sleep.
   - If $r = p - 1$, $r \leftarrow 0$, the thread recomputes the mean value of each cluster based on the local intermediate data of all the threads, then reinitializes the local intermediate data of each thread. If the algorithm does not converge after the current iteration, this thread sends a broadcast single to wake up all the threads. Go to Step 3 to start a new iteration.
5. Join the threads. Replace the value of each data with the mean value of the cluster which the data belongs to.

In this problem, the input data is a **1024×1024** matrix which is stored in the 'input.raw'; the value of each matrix element ranges from 0 to 255. We will have **6** clusters ($K$=6). The initial mean values for the clusters are 0, 65, 100, 125, 190 and 255, respectively. To simplify the implementation, you do not need check the convergence; run **50** iterations and output the matrix into the file named 'output.raw'.

- Name the program as p2.c. Pass the number of threads $p$ as a command line parameter
- In your report, you need report the execution time for the **50** iterations (excluding the read/write time) for $p = 4, 8$. Compare the execution time with the version which you implemented in PHW 2, discuss your observation in your report.

# 3 Submission

You may discuss the algorithms. However, the programs have to be written individually. Submit the code (p1a.c, p1b.c, p2.c) and the report via **Blackboard**. The report is preferred to be written in PDF. Your program should be written in C or C++. Make sure your program is runnable. Write clearly how to compile and run your code in the report as well. It is recommended to include a Makefile [2] along with your submission. If your program has error when we compile or run your program, you will lose at least 50% of credits.

# References

[1] "Command Line Parameter Parsing,"
http://www.codingunit.com/c-tutorial-command-line-parameter-parsing

[2] "Using make and writing Makefiles,"
http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html