91.515 Operating Systems I   Proj #1 September 1, 2016

1. The **due date** for this assignment is Thursday, September 29.

2. Submission details will be provided in class, but you will be required to include your **source code**, your **output** and a **write-up** of your results described below.

   If you have done this project before (as part of my COMP.3080 undergraduate course) you must redo the assignment using either the Win32 System Services API on a Windows platform, or using the POSIX semaphore and shared memory APIs on a Linux/UNIX system (see Linux man pages for sem_overview (7) and shm_overview(7).

3. The problem you must solve has been described in class and is formalized as follows:

   The problem is a **producer-consumer** problem requiring a **single producer** (of donuts) process, and an **arbitrary number of** consumer processes. The producer must: **create a shared memory segment** with 4 ring buffers and their required control variables, one for each of four kinds of donuts

   You must **create semaphores** for each ring buffer to provide mutually exclusive access to each buffer after creating and mapping the segment.

   The producer will enter an endless loop of calling a **random number** between **0 and 3** (where each value represents one of 4 possible donut flavors) and then producing one donut of the selected flavor corresponding to the random number (remember that the producer could get blocked if it produces a donut for a ring buffer which is currently full). A donut can be thought of as an integer value placed in the ring buffer, where the integer is the sequence number of that type of donut (i.e. the initial entries in each ring buffer would be the integers 1 to n, where each integer represents an instance of a donut of the flavor found in that buffer, and where n is the size of the buffer. When the 1st donut of some particular flavor is removed by a consumer from the first slot of the buffer holding that flavor, the nth+1 donut of that flavor (integer n+1) can be placed into the newly vacated slot in the buffer holding that flavor when it becomes available from the producer).

   Consumers are started **after the producer** (any number may be started at any time after the producer) and each consumer must find the **shared memory ID** created by the producer and **attach the segment** to their address spaces. Each consumer must:

   get and attach the shared memory segment which contains the ring buffers and corresponding shared out-pointers

   get and use the semaphores created by the producer to coordinate access to each of the different flavor ring buffers

enter a loop of some number of **dozens of iterations** and begin **collecting dozens of mixed donuts** using a **random number** (as does the producer) to identify each donut selected

each time a specific consumer **completes a selection of a dozen donuts**, that consumer process makes an entry in a **local file** (one file per consumer; use your own naming convention) as follows:

```
consumer process PID: 34567       time: 10:22:36.344    dozen #: 4

plain   jelly   coconut   honey-dip
11        3        9         15
38        7       20
         11       24
         19       30
         24
```

4. Your output for submission should include a catenation of each of the local files of each consumer, **in the order that you started your consumers**

5. Your write-up should include a statement as to how much success you felt you had with this project, and your **write-up** must provide a **quantitative discussion of your results and any observations or problems you encountered in the project.** For this problem, it is your write-up which will determine your grade. Writing code to implement the assignment is step one, but experimenting with different configurations and providing your results and conclusions in your write-up is most important.

6. You must run a test set with **1 producer** and **5 consumers**, where each consumer runs until it collects **10 dozen donuts** using a queue size of **50 slots** for each flavor ring buffer, and submit these complete results (since the probability of deadlock in such a configuration is known to be low, you should be able to get results here with one or two tries … you'll need a full run to generate all the consumers' output files). Once you've generated results for this fixed configuration, you must experiment with the **queue depth setting** in additional runs (however many more you think you need) to collect enough data to allow you to **produce a graph** of the **probability of deadlock vs. the depth of the queues** for the 5 consumer, 1 producer case. Is this distribution **linear or non-linear** ? Summarize and discuss these results, your initial results, and any other observations you've made in your write-up (don't include the consumer output files for these additional runs, just summarize what you found and include your graph).

7. From step 6, you must find the **50% deadlock queue size** and use this specific queue size to generate a **second deadlock probability distribution** that varies the number of consumers **from 1 to 10** using the fixed queue size previously determined. Prepare a **graph for these results** and discuss these results in your write-up (don't include the consumer output files for these runs either, just summarize what you found and include your graph).

8. You will find a system call help file to assist you in using UNIX system calls at the URL: **http://www.cs.uml.edu/~bill/cs515/call_help_assign1.txt**
and for the Windows Win32 API at the URL:
http://www.cs.uml.edu/~bill/cs515/ Win32_calls_Assgn2_and_A1.htm