**CSCI 331 Project Two**

**Note: This is a project assignment from your zybooks textbook**

**Project objective**

Implement the solution to the bounded buffer problem from the section titled Semaphores, but without any P or V operations. Observe and eliminate a race condition.

**Description**

* The buffer is a large array of n integers, initialized to all zeros.
* The producer and the consumer are separate concurrent threads in a process.
* The producer executes short bursts of random duration. During each burst of length k1, the producer adds a 1 to the next k1 slots of the buffer, modulo n.
* The consumer also executes short bursts of random duration. During each burst of length k2, the consumer reads the next k2 slots and resets each to 0.
  + If any slot contains a number greater than 1, then a race condition has been detected: The consumer was unable to keep up and thus the producer has added a 1 to a slot that has not yet been reset.
  + If any slot that consumer reads contains a number 0, then a race condition has been detected: The producer was unable to keep up and thus the consumer try to read data in a slot that has not yet been added.
* Both producer and consumer sleep periodically for random time intervals to emulate unpredictable execution speeds.

|  |  |
| --- | --- |
| producer thread:  while (1)  get random number k1  for i from 0 to k1-1  buffer[(next\_in + i) mod n] = 1  next\_in = (next\_in + k1) mod n  get random number t1  sleep for t1 seconds | consumer thread:  while(1)  get random number t2  sleep for t2 seconds  get random number k2  for i from 0 to k2-1  data = buffer[(next\_out + i) mod n]  if (data > 1)  exit and report race condition that consumer too slow  else if (data == 0)  exit and report race condition that producer too slow  else  buffer[(next\_out + i) mod n] = 0  next\_out = next\_out + k2 mod n |

**Assignment**

1. Experiment with different values of n, k, and t until a race condition is observed.
2. Modify the solution by including the necessary P and V operations in the code.  
   If general P and V operations are not provided by the thread library then first implement P and V using binary semaphores (mutex lock or spin locks.)