Due date: 10/02/2015

1. Implement producer-consumer problem using only semaphores and where "item" is a simple `int`.

```
1  //producer-consumer algorithm;
2  #define N 100 //number of slots in the buffer;
3  semaphore mutex = 1;
4  semaphore emptySlot = N; //counts empty slots in the buffer;
5  semaphore fullSlot = 0; //counts full buffer slots;
6
7  int buffer[N]; //make a buffer with size N;
8  int produced_Items; //number of items produced;
9  int item;
10
11 void producer () {
12     while (TURE) {
13         item = produce_item(); //generate something;
14         //if there are no empty slots, wait; else, decrement empty slot;
15         wait ( &emptySlot);
16         //if another thread are using the buffer, wait; else, enter critical region;
17         wait ( &mutex);
18         //CRITICAL REGION. ONLY ONE THREAD COULD HAVE ACCESS TO. Put new item in the buffer;
19         insert_item (item);
20         signal (&mutex); //leave critical region, release the buffer;
21         signal (&fullSlot); //increment number of full slots;
22     }
23     return NULL;
24 }
25
26
27 void consumer (){
28     while (TRUE) {
29         wait (&fullSlot); //decrement full count;
30         wait (&mutex); //enter critical region;
31         item = remove_item(); //CRITICAL REGION. Take item out from buffer;
32         signal (&mutex); //leave critical region;
33         signal (&emptySlot); //increment number of empty slots;
34         consume_item(item); //do something with the item;
35     }
36     return NULL;
37 }
```

Reference:[1,2]

[1]Book P131 Figure 2-28.

[2]`http://stackoverflow.com/questions/19844716/producer-consumer-program-using-semaphores-and-pthreads`

2. What is the difference between the producer-consumer problem above and the reader-writers problem[3]?

Both producer-consumer algorithm and reader-writers algorithm are used when multiple threads need to access a shared resource/buffer; in other words, to solve concurrency problems.
The differences:
1). In the readers-writers problem, at any point, only one writer process can be inside the critical section; which shows in line 28 and line 30, `wait(&db)` and `signal(&db)` is the `semaphore db` that separates other writers come to the critical region.
2). In the readers-writers problem, writers could be the first to access the critical region. However, in the producer-consumer problem, only producer could first access the critical region because the `semaphore full` initializes to 0. In line 29, `wait(&fullSlot)` will block consumer to go for futher steps unless producer goes first.
3). In the readers-writers problem, more than one reader threads may simultaneously access the resources and `read_data_base()` in line 14. For example, if a second reader thread want to `read_data_base()`, it could actually do so because the `if` statement on line 10 keeps `db` from spinning and allows second, third thread threads to jump to line 13 and `read_data_base()`. However, in the producer-consumer problem, only one thread could access the critical section due to `wait(&mutex)` and `signal(&mutex)` on line 17 and 19.

```
1  //readers-writers algorithm;
2  semaphore mutex = 1;
3  semaphore db = 1;
4  int rc = 0; //number of processes reading the data;
5
6  void reader () {
7      while (TRUE) {
8          wait (&mutex); //get access to critical section;
9          rc = rc + 1; //one more reader;
10         if ( rc == 1) { //first reader;
11             wait (&db);
12         }
13         signal (&mutex); //release critical section;
14         read_data_base();
15         wait (&mutex); //get exclusive access to critical section;
16         rc = rc - 1; //fewer reader;
17         if ( rc == 0 ) { //last reader;
18             signal (&db);
19         }
20         signal (&mutex); //release exclusive access to critical section;
21         use_data_read();
22     }
23 }
24
25 void writer() {
26     while (TRUE) {
27         think_up_data();
28         wait (&db); //exclusive access to critical section;
29         write_data_base();
30         signal (&db); //release exclusive access to critical section;
31     }
32 }
```

[3]Book P171 Figure 2-48.

Reference:[4,5]

3. Prove the following property from *bakery algorithm*: If P_i is in its critical section and P_k (k != i) has already chosen its `number[k]!= 0`, then`(number [i], i) <(number[k], k)`.

The central idea for the bakery algorithm is to name processors $1, ..., N$. If two processors choose the same number, then the one with the lowest name goes first.
During the execution of statement `while (number [i], i) <(number[k], k)`, processor i read the current value of $number[k]$. Thus, `(number [i], i) <(number[k], k)` statement must be true.

Reference:[6]

4. Show that, if `wait()` and `signal()` semaphore operations are not executed atomically, then mutual exclusion may be violated.

First, `wait` and `signal` semaphore must be executed atomically, meaning only one thread can be executing the code at one time. If both threads are executing `wait`, then it is possible that both operations are proceeding to decrement the semaphore value, and both threads are proceeding to enter to the critical region, thus violates the mutual exclusion rule.

5. Modify the *sleeping barber*[7] solution so that it is clear which customer's hair gets cur next. The program should display the number of the next customer to get his/her hair cut.

This algorithm is correct. The algorithm first need to go through `customer()` since in the `barber()`, `custReady = 0`, so that the barber is still asleep. In the `customer()` section, after the `signal(custReady)` statement, `custReady` are being implemented 1 (from 0 to 1), thus, `barber()` section could run. At this point, in the `customer()` section, since `barberReady` = 0 (line 24), thus the `customer()` thread could not go until `barber()` thread goes. At this point, line 13: `signal(barberReady)` is being executed and in line 14, executes `signal(accessWRSeats)`, which means barber enters into the critical section and cuts hair. The `cout` statement under `barber` section could indicates the first, second etc customers get hair cut based on `numberOfFreeWRSeats` available.

```
1  # sleeping barber solution
2  # The first two are mutexes (only 0 or 1 possible)
3  Semaphore barberReady = 0
4  Semaphore accessWRSeats = 1      # if 1, the # of seats in the waiting room can be incremented o
5  Semaphore custReady = 0          # the number of customers currently in the waiting room, ready
6  int numberOfFreeWRSeats = N      # total number of seats in the waiting room
7
8  def Barber():
9    while true:                    # Run in an infinite loop.
10     wait(custReady)              # Try to acquire a customer - if none is available, go to sleep
11     wait(accessWRSeats)          # Awake - try to get access to modify # of available seats, oth
12     numberOfFreeWRSeats += 1     # One waiting room chair becomes free.
13     signal(barberReady)          # I am ready to cut.
14     signal(accessWRSeats)        # Don't need the lock on the chairs anymore.
15     #Cut hair here.
16     cout numberOfFreeWRSeats
```

[4] http://www.codingdevil.com/2014/04/c-program-for-reader-writer-problem.html
[5] http://www.cs.uic.edu/~rchitra1/problems.html
[6] http://research.microsoft.com/en-us/um/people/lamport/pubs/bakery.pdf
[7] https://en.wikipedia.org/wiki/Sleeping_barber_problem

```
17
18 def Customer():
19     wait(accessWRSeats)          # Try to get access to the waiting room chairs.
20     if numberOfFreeWRSeats > 0: # If there are any free seats:
21       numberOfFreeWRSeats -= 1  #   sit down in a chair
22       signal(custReady)         #   notify the barber, who's waiting until there is a customer
23       signal(accessWRSeats)     #   don't need to lock the chairs anymore
24       wait(barberReady)         #   wait until the barber is ready
25
26     else:                       # otherwise, there are no free seats; tough luck --
27       signal(accessWRSeats)     #   but don't forget to release the lock on the seats!
28       # (Leave without a haircut.)
```

Reference: [8]

I have abided by the Wheaton Honor Code in this work.

[8] http://brahms.emu.edu.tr/rza/comp483week4.pdf