Name: Manita Ngarmpaiboonsombat
Student ID: 01940831
Email: manita_ngarmpaiboonsombat@student.uml.edu

Assignment#3

In my opinion, my degree of success for this assignment is 75%.
I was able to successfully run all of the example code provided by the professor, including the sequential full connection and the communication between the buffer manager and the local producer/consumer. I totally understand the whole concept of how they work together, however I cannot successfully implement the code and do the experiment.

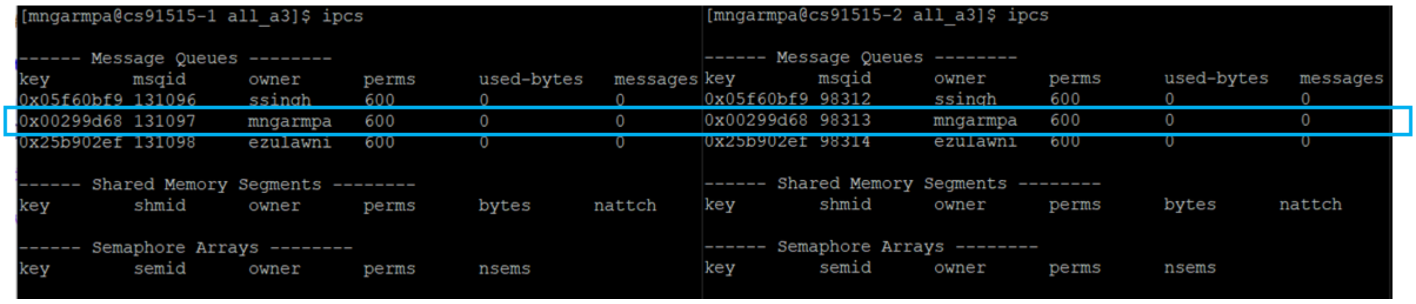
This shows how I grade my success at 75%

Understand the whole concept	25%/25%
Test by successfully running your example code	25%/25%
Try to adjust the code	25%/25%

- Build a thread for handling a message queue.
- Integrate the message IPC into the node controller and run it by a thread which creates a message queue.
- Integrate the message IPC into the local producer/consumer code and ask for permission from the node controller to connect with the buffer manager.

Completely running the code and doing the experiment	0/25%

I made the node_controller.c, bug_mgr_server.c, net_prodecer.c, net_consumer.c, utilities.c, msg_util.c, bug_mgr.h, ddonuts.h and Makefile. I can compile them without any compilation error. I also created node1.sh, node2.sh and node3.sh for local producers and consumers which can include 6 producers and 16 consumers, according to the instructions.
Screenshot shows the message queue that is created in each node after the full connection.



```
[mngarmpa@cs91515-1 all_a3]$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x05f60bf9 131096     ssingh     600        0            0
0x00299d68 131097     mngarmpa   600        0            0
0x25b902ef 131098     ezulawni   600        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

```
[mngarmpa@cs91515-2 all_a3]$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x05f60bf9 98312      ssingh     600        0            0
0x00299d68 98313      mngarmpa   600        0            0
0x25b902ef 98314      ezulawni   600        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

Below are the concepts that I understood:

<u>Node controller</u>

1. Fully node connection (has to be sequential connect if using example code)
    a. Listen for all connections
    b. Connect to the listening nodes and accept some connections
    c. Connect to all listening nodes
2. Lamport's algorithm
    a. Building at least 8 request queues (Linked list)
    b. Building a thread
        i. The thread will handle a message queue to communicate with local producers and consumers.
        ii. The thread will set up a receive request, then wait. When it receives the msg type, it forms a lamport REQUEST message to produce/consume a donut and put it in an appropriate request queue which is sorted by the timestamp then node-id.
        iii. It will send the REQUEST message out to all peer NCs.
        iv. When it receives a REPLY back from all peer NCs and its request is on the head of the queue, it would do the CS.
        v. After finishing the CS, it will remove the request from the queue and then send the RELEASES message to others.
        vi. It will notify the next client when it is their turn and listen from the message queue again.
        vii. Node controller will be terminated by signal handler in order to clean up the message queue

<u>Buffer Manager</u>

1. After getting a request from the producer/consumer, the buffer manager will respond to the child. Then, the child will handle the request by figuring out which donut flavour to produce/consume (using a serial number in shared memory to get the donut number and reply to the producer/consumer).
2. The child will evaporate and close the connection.
3. All connections between the buffer manager and producer/consumer are a single transaction.

<u>Producer and Consumer</u>

1. Using node id and port id to talk with buffer manager
2. Producer/Consumer requests permission from the node controller to connect with the buffer manager. If they get permission, they can directly connect to the buffer manager.
3. Then the node controller will read a message from the client and notify the queue to be ready.