Project 2 FAQ

1.) Semaphore usage;

Semaphore(char* debugName, int initialValue)

**Note initialValue is the number of resources available (i.e. number of times Wait() is called before putting the thread to sleep)

Because semaphores in NachOS do not have a default constructor a single pointer implementation of an array will cause errors. So, use double pointers.

```
/* this is how to declare a double-pointer array; it is an array of pointers; the rest of the above example code is still valid. */

Semaphore **mySemaphores;

// Initializing the array of Semaphore pointers in the array:

mySemaphores = new Semaphore*[100];
   /* This is now an array of semaphore pointers; where each pointer has not been initialized */

      for (int i = 0; i < 100; i++)
      {
            mySemaphores[i] = new Semaphore("Debugging Name", 1);

           // Each of the 100 elements in this array will now be initialized to a value of 1.
      }
```

2.) Are we allowed to implement the Lock and Condition classes in synch.cc (with interrupts rather than semaphores)?

****Note: only relevant to Task 1 (Non-semaphore DP)**

Changing them might break some of the guts of Nachos that depend on the stubs, in future assignments. Feel free to make a function or class that handles that, if you like, or a macro instead.

3.) Does # of meals need to be >= to # of philosophers?

No. Your program should handle any and all kinds of input. It is your system though, and as long as it is within the specs provided for the project you can handle these inputs how you want.

2 example edge cases and example solutions. (These are far from the only 2 so continue to think and take into account other cases):

P=1, M=1 Not possible as you need 2 chopsticks to eat.
Solution: End the program with the proper error message

P=2, M=1 Possible but only one philosopher will eat and there is possible deadlock if each philosopher picks up a chopstick

Possible solution: Wait some time then ask one philosopher to drop his/her chopstick so the other can continue

4.) Can we use busy waiting loops for etiquette on Task 2?

For task 2-4 if there is a common resource to be handled you must use a semaphore. So if you have a vairable that is **edited** by mulitple threads use a semaphore, if they simply read the variable there is no need to protect it with a semaphore.

5.) I have some global counters for task 2 to keep track of the amount of meals left and how many threads have finished. Would it be good practice to use semaphores to protect these variables for incrementing/decrementing? The idea is if this were a multi-threaded device, then potentially multiples threads could attempt to alter these variables at the same time causing issues... Or am I going a bit too far with trying to synchronize things?

Yes, if there is any variable that is accessed by multiple threads, then you need to protect that using a semaphore. But, if the variable is used with another variable that is protected using semaphore than you may not need to add another semaphore.

6.) What constitutes an eaten meal for Dining Philosphers?

A philosopher will have finished eating 1 meal after his/her 3-6 cycles of eating.

7.) "As with Task 1, make the philosophers yield between attempting to pick up the left and right chopsticks. Run the same tests used in Task 1 and record the results. Were the results the same or different? Why? Undo any changes made to accommodate this question before submitting your assignment."
Does this mean note the differences in the task 1 and task 2 output, both with the same parameters and yields before picking up the chopsticks?

You would already have output for task 1 as per task1 q2. Here, you need to use same parameter as you used in task 1. You can them compare results between task 1 and task 2 when using yield. You can also discuss results between task 2 with or without use of yield.

8.) If a person is reading their mailbox can others still leave mail for them while they are in the reading loop?

Yes. Others should still be able to leave the reader messages. The reader must still call yield after each message is read.

9.) I was looking at the sample output posted on Moodle and we saw that all everyone enters the post office at the same time in the beginning. Should everyone still yield for the next thread/person even when their mailbox is empty in the beginning?

Yes

10.) This section of the output is confusing me.

- Person 2 enters the post office
- -Person 2 checks - he has 0 letters in mailbox.
- -------Person2's mailbox is empty.
- --------Person 2 is trying sending Pattern3 to Person 1
- ---------Person 2 tries to claim freeSpaceSemaphore[1].
- ----------Person 2 gets and decreases freeSpaceSemaphore[1] by 1
- -----------Person 2 claims mailboxSemaphore[1]
- ------------Person 2 successfully send Pattern3 to Person 1
- -------------Person 2 successfully updates the overall msg count to 2
- Person 3 enters the post office
- -Person 3 checks - he has 1 letters in mailbox.

Why didn't Person 2 leave the post office? Referring to the algorithm, after placing a message in the mailbox, they leave the post office, wait for 3 to 7 cycles, and then re-enter the post office. Person 2 ends up leaving a bit later. Is there a reason for this? Perhaps I'm misunderstanding something crucial?

Person 2 does leave in their next step a few lines down, and then starts to do their wait cycles right after, so it is still following the algorithm.

11.) Also, when it comes to yielding,

- ----------Person 1 yields the 1th cycles
- ----------Person 3 yields the 2th cycles
- ----------Person 3 yields the 3th cycles
- ----------Person 1 yields the 2th cycles

The word cycle is used here. As in yield, and go the end of the ready queue until it comes back again? That's a cycle? If that's the case, how does Person 3 yield twice in a row?

Note that the output was run with an rs seed, so there are going to be random yields thrown in that will account for threads not going in the order they would if they only yielded when you called yield. If a single thread starts to do things in a weird order, then that would be a different issue that would be a problem.

12.) How do you represent mailboxes for Task 3?

To represent a mailbox, you could simply make use of a multidimensional array of characters as storage. Sending a message means setting the value of the array to the message, and reading means printing the message and clearing the array.

For example,
char*** mailbox = new char**[P]; //This will create a 3-dimensional array of characters. The first dimension is the number of people, the second dimension will be the number of slots and the third dimension is the actual message.

```
// Now we allocate space for each person's mailboxes
for (int i=0; i<P; i++) {
        mailbox[i] = new char*[S];
        // Now allocate memory for the messages
        for (int j=0; j<S; j++) {
                mailbox[i][j] = new char[some_large_number]
                // Make sure this is big enough to hold any message you put there
        }
}
```

13.) After a person fails to send a piece of mail to another person three times due to the recipient's mailbox being full, are we allowed to discard the undelivered message and let them send mail to someone else? Or should the deadlock solution ensure that every piece of mail gets delivered and no mail is deleted?

No mail should be deleted and all mail should be sent at some point. There are many solutions for dealing with this. Feel free to become creative.

14.) I apologize for asking this, but how do we create reader threads?

You can simply do it like as:

```
Thread *reader;
for(int i=0; i< r; i++)
{
reader = new Thread("Reader Thread");
reader->Fork(read, 1);
}
```

15.) What can we use to represent latency of reading/writing?

You can use busy waiting loops with Yield(). If you represent the file as some array, looping through the array would simulate latency of reading a file and changing one or more values of the array could represent writing.

Depending on how you represent latency a reader may finish reading before N readers get to the file, THIS IS OKAY. So long as the pattern of N readers → 1 writer → N readers → 1 writer is maintained.

TAs advice on how to represent eating, thinking, reading, writing

Some useful suggestions

- You can use yield to represent eating or thinking or reading or writing. Only for handling critical resources, you need to use semaphore if that is the part of task.
- You should not need to have exact output as in sample outputs. The output depends on when you print them. It may differ on how you implement. The only thing you need to make sure is the core idea. For example, in readers writers problem you just need to maintain N readers and 1 writer pattern. It does not matter how the readers among them reads and finishes.
- Keep the output clean and clear. Remove unnecessary outputs but make sure you print all ids that represents thread so it is clear which thread is doing what.

Number of threads: a TA's advice on Minimum Number of Philosophers and Post Office People

- Lots of people have had max threads of 100 or 10 in Assignment 1, this is far too few.
- For Assignment 2, you must handle 10k threads for full credit. That is Philosophers or Mailbox Readers.
- For the PO Box, the max size should be 1k at least, preferably 10k. If you can't get 10k, have a good reason why not.
- Meals and Messages should also get to 10k.
- Be sure to test all sorts of conditions such as 10k 10k 10k, or 10k 2 10k, etc. Find pinch points and try to break it, then fix it.

TA's General Advice:

- For semaphore post office, you do need to check your anti-deadlock prevention strategy in a loop (3 times).
- For anything using semaphores, it is best to ensure you aren't yielding or waiting on semaphores while holding mutexes (unless it is vital that no other thread gets past that mutex while you wait, which is unlikely). e.g. don't hold a mutex on the number of messages sent while waiting for a mailbox to clear.
- Ensure it compiles.
- Ensure you answer all questions in the report completely, and ensure your output meets the minimum requirements.
- When your Philosophers leave at the end, there should be some kind of report about how much each has eaten, and the total meals that were eaten. Same thing for the post office. When all is done, and everyone is leaving, have them each report how many letters are in their mailbox and how many messages they sent/received, and how much this adds up to.
- In the post office there are shared resources such as the number of messages, and any other global thing that two threads can access. Basically in task 4 you use a counting semaphore for the mailbox, in addition to all the binary semaphores you would need for mutexes.