Amy Canelas
C00416506
Project 1

Task 1

Try # 1	# of philosophers	# of meals	runtime in milliseconds
test run 1	2	2	2950.914628 ms
test run 2	4	7	3910.708149 ms
test run 3	10	7	4977.750536 ms

Make the philosophers yield between attempting to pick up the left and right chopsticks. Run the same tests you ran in the previous question, and record the results. Were the results the same or different? Why? Undo any changes made to accommodate this question before submitting your assignment.

- Results were different, whenever I yielded between chopsticks it raised my milliseconds by quite a bit. It must've gone up since we were forcing the program to halt in between checking for chopstick permit availability.

Try # 2	# of philosophers	# of meals	runtime in milliseconds
test run 1	2	2	30238.287166 ms
test run 2	4	7	4022.55725 ms
test run 3	10	7	3194.211649 ms

Task 2

Did you experience any deadlock when testing this task? How was it different from Task 1?

- I only finished up to the point where messages were being delivered. It didn't involve the usage of a barrier since it didn't require threads to start and end at the same time. This task used a 2D array to keep track of people in the simulation as well as the number of mail slots per person. There were a lot more methods to go through since we were simulating a post office.

Task 3

How did this task differ from the previous synchronization tasks?

- It felt more straight forward when it came to implementing the pseudocode. I felt that by adding a string attribute to each type of thread (whether reader or writer) within Main before creating my ReaderWriter object helped simulate a cleaner program when trying to run the threads in my ReaderWriter class. It also didn't involve any deadlock situations, since the pseudocode provided handled what we were trying to achieve within our scope. At least I believed the issue that I wasn't able to print my pattern had nothing to do with deadlock, but more to do with me not being able to write the correct conditions.

What kind of problems do you see when N is very large (i.e. high priority is given to readers)?

- Writer starvation, perhaps. Many readers will be accessing the buffer since multiple of them can do so, meanwhile the writers will access one at a time therefore have a longer waiting time before acquiring space in the buffer.

Continuation of Report

In your own words, explain how you implemented each task. Did you encounter any bugs? If so, how did you fix them? If you failed to complete any tasks, list them here and briefly explain why.

Overall, I encountered bugs in each task. I am under the impression task 1 works, task 2 has a semi working algorithm and task 3 works but does not output the correct pattern.

- Task 1: Honestly, I just tried to program intuitively as to how I believed scenarios would happen between philosophers. I went to the TAs office a lot for this specific problem. I decided to opt to use the method where even philosophers choose right before left and odd philosophers choose left before right.
    - Fixable bugs:
        - I didn't realize we needed a second barrier so that philosophers could leave at the same time towards the end.
- Task 2: This one was the hardest. I left it for the last one since I assumed I would do better if I went by implementing a 2D array instead of creating a Mailbox class. I couldn't see how you could access Thread commands within Main even after extending Thread, so I opted for a solution where I used both. I created a Person and Mailbox class where Person was the class designated to run the thread code, and Mailbox was the class being called to execute the simulated tasks.
    - Fixable bugs:
        - I couldn't figure out how to choose a j index when iterating through the array. At first I thought having [j + 1 % totalSlots] would work because it would always be 0 through (totalSlots-1), but then I opted to save an integer with the value of mySemaphore.availablePermits() - 1 (which gave you the same range but it was less finicky).
- Task 3: I liked this one the most although it gave me trouble finding a way to make the pattern happen. When I originally was trying to solve for a pattern, I decided to build it from Main and it worked. Yet when I tried to sync up my nicely ordered threads, they wouldn't behave the way I wanted with the semaphores in my ReaderWriter class. In the end, I gave up and simply opted to have them display the default input of all reader threads reading and then writers after.
    - Fixable bugs:
        - My output kept projecting that my writer threads were writing at the same time, but then I used a loop with a yield() for the writing simulation which helped display the correct behavior.

What data structures and algorithms did you use for each task

- Task 1: booleans, integers (dining philosopher)
- Task 2: 2D arrays, booleans (producer consumer)
- Task 3: integers (readers-writers)