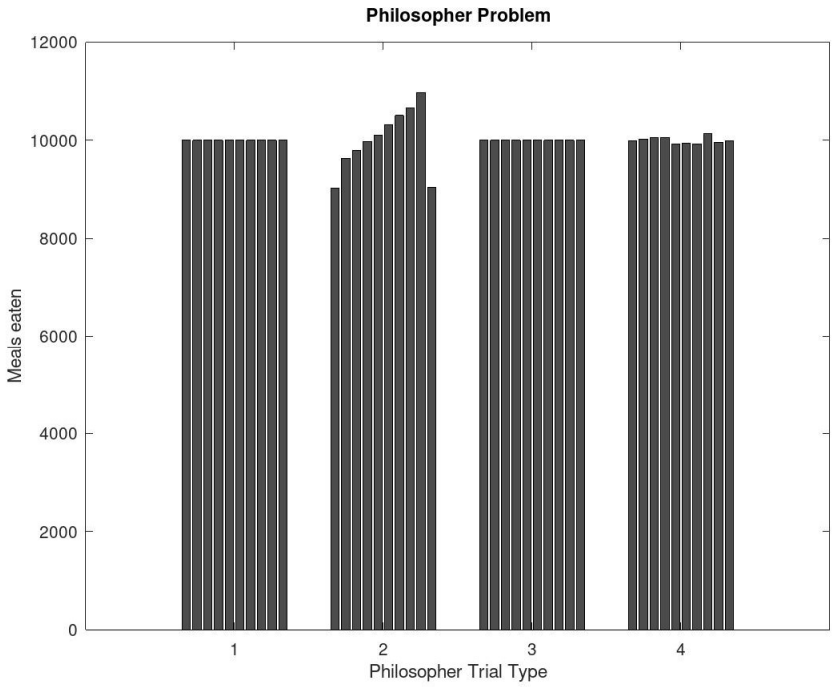
Assignment 8 Writeup

Create your own histogram of how often each philosopher gets to eat with respect to each algorithm. In your histogram, CLEARLY indicate which bars correspond to each of the four programs you wrote. Include your histogram in the PDF. Clearly indicate the meanings of the x and y axes and indicate which bars correspond to each of the four algorithm/problem pairs you coded. (5 points)



Each of the numbers correspond to a trial that was run 1 : standard_arbiter, 2 : standard_rh, 3 : random_arbiter, 4 : random_rh.

Do you think that it is possible to prove that none of these solutions deadlock with the experimental data you collected? Why or why not? If you think you can, why do you think the data is sufficient. If you think you cannot, then what COULD you do to offer such a proof? (5 points)

It is possible to prove that none of them deadlock if all the meals of all the philosophers add up to the number of meals. That way you know that all the meals where eaten with none left at the table because of a deadlock.

What would you need to do to establish that the arbiter (waiter) solution to this problem is FREE of starvation? Starvation occurs when one or more threads never get to run in their critical sections. In the example problem, it would correspond to a philosopher never being allowed to eat. How might you go about proving that the arbiter solution is STARVATION FREE. You don't need to provide a formal proof, but do provide a discussion of what facts you would need to establish to prove prevention of starvation. Hint: This might consider thinking about how the mutex operators themselves are implemented along with the mechanics of the arbiter solution (5 points)

The arbiter solution of the philosopher problem is the most fair in which all of the philosophers get to eat an even amount of meals since each one of them get a turn eating. It is possible to know if it is free of starvation if all of them got at least one meal and also if the meals are spread out among the philosophers mostly evenly so there are no long periods of starvation. Due to the solution of the arbiter, there is a lot of waiting in between philosophers getting meals since each one of them get a

turn. It has to wait for its turn again so it can eat. Not efficient but the most fair.

Once you've identified which of your four programs was unfair, examine the code and speculate on how it is interacting with the allocation of resources (chopsticks) to create the unfairness. In other words, explain why philosophers with lower ID numbers seem to get preferential treatment. Again, a formal proof is not necessary, but you should at the very least show, by discussion, a few examples of how the unfairness arises (5 points).

The most unfair program of my bunch was the second one or standard. I think this is because they are assigned chopsticks and they have to be available to get those chopsticks. In my case it is the reverse, philosophers with higher ids get more priority. I think this is there is always going to be one chopstick available at the top because the way it is designed. So the rest of them have to wait for them to eat before anyone else can eat. So when they are done eating it goes down the line. If the philosopher who ate first finishes before the last one gets a turn then it will be skipped for that cycle.