Dustin Rowland Travis Derouen Micah Riggan Brandin Jefferson

dcr3891 tjd5951 mlr9790 bej0843

**Report for Stage Two: Soup and Salad**

**Task 1 (dining with waiting):**

1. When implementing and testing your solution, did you notice any deadlock? Deadlock occurs

when threads cannot progress due to improperly controlled access to critical resources. (In this case, the critical resources are the chopsticks.) How did you solve any deadlock problems?

One deadlock situation that occurred was if all the philosophers were to pick up their left chopstick. This would leave all philosophers waiting on their right chopstick which was being used. We solved this problem by making a philosopher pick up his right chopstick as soon as he picked up his left one. This allowed the at least the first philosopher to acquire both chopsticks and eat which would break the chain in the case stated above.

Another deadlock situation that could occur is if there was only 1 philosopher. This was solved by adding another chopstick to allow him to eat however many meals needed.

2. Make the philosophers yield between attempting to pick up the left and right chopsticks. Run at least 3 tests with various parameters and rs seeds. Record your findings and explain your results. Undo any changes made to accommodate this question before submitting your assignment.

In every case, deadlock occurred. Each philosopher would enter, then once all entered, they would all sit. After everyone was seated, the first philosopher would pick up his left chopstick and then yield. This would let the next philosopher pick up his left chopstick and then yield. Eventually every philosopher picked up their respective left chopstick causing all the philosophers to wait for an available chopstick. But since every philosopher was waiting on a chopstick, none of them were able to eat and put down their chopsticks. So this caused all philosophers to wait forever.

**Task 2 (dining with semaphores):**

1. Run your solutions to Task 1 and Task 2 with the same parameters, including rs

seeds. Note the number of ticks that NachOS runs. Do this for at least 3 different sets of parameters. Record your results in a table, including rs seeds, number of philosophers, and number of meals. Explain your findings.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Philosophers** | **Meals** | **Seed** | **Ticks for Task 1** | **Ticks for Task 2** |
| 3 | 3 | 3940587 | 714 | 786 |
| 10 | 20 | 1234 | 3119 | 4052 |
| 300 | 300 | 387182 | 126,349 | 95,650 |
| 500 | 500 | 456798 | 300,760 | 159,745 |

It seems that with less data to process and less congestion, busy waiting loops outperform semaphores. However, with more data and more congestion, semaphores seem significantly more efficient. Initially, we only tried smaller numbers for the philosopher and meal counts and noticed that busy waiting loops came in first each time. We thought that since different group members did task 1 and task 2 that perhaps task 1 had more efficient code. However, upon a little bit of research and experiments with higher numbers for philosophers and meal counts, we saw that semaphores do seem to have an edge over busy waiting loops in larger data sets. This can likely be attributed to large amounts of wasted “waiting” cycles in the busy waiting loop implementation that do not occur in the semaphore implementation.

2. 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.

The same result occurred from Task 2. The yield statement after a philosopher picks up his left chopstick causes another philosopher to pick up his chopstick and yield. Each philosopher will yield after picking up the left chopstick and no philosophers will have a right chopstick free. Thus, they are all stuck in a deadlock as in Task 2.

**Task 3 (messaging with waiting):**

1. Explain the method you used to resolve the deadlock problem. Why did you choose this particular method?

For the deadlock in task 3 we implemented a countdown of 3 tries. If it fails to put the message after 3 tries then it just destroys the message. We chose this method because it still allowed for a message to be sent to a mailbox if it became not full soon. For example, if a message box was full, the sender would wait for a bit before attempting to send again in case the receiver had read a message. Also, this method let us not waste a lot of time in waiting for space to send the message.

**Task 4 (messaging with semaphores:**

1. Did you experience any deadlock when testing this task? How was it different from Task 3?

This task still had the issue of having the possibility of sending a message to a full mail box. We solved this issue by attempting to send the message 3 times. After 3 times, the message was thrown away if it was not successfully added to the messagebox.

**Task 6:**

1. 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.

For the busy waiting loop Philosopher solution I tried to model the stages of the algorithm and then I used busy wait loops to handle picking up the chopsticks while checking to make sure they didn’t eat more food than what existed. I had problems getting all the philosophers to enter at the same and to sit down together. I also had trouble getting them to exit when everyone was done, but I solved this by using a flag and looping through every philosopher to make sure they were on the finished thinking stage while all the meals were gone. Once they were all on this stage I incremented the stage by one and had them all leave. For the semaphore section of philosophers I just changed all the busy waiting loops to use semaphores.

2. What sort of data structures and algorithms did you use for each task?

We implemented various flag algorithms and a Philosopher’s struct and a data structure for the messages. We used multiple busy waiting loops and multiple semaphores. We also implemented a global flag, that is used to coordinate sitting and entering of philosophers.