Figure 6.15, p. 236, outlines a partial solution to the Dining Philosophers Problem for 5 philosophers. However, that solution may deadlock. The text presents three possible modifications, pp. 235-236 (the bullet items), that prevent deadlock. Each solution requires an array of semaphores called chopstick (p. 235). Other semaphores may also be required. Choose one of the following solutions and integrate it into the code of Figure 6.15:

1. Allow at most N-1 philosophers to eat at any one time. This can be done with a counting semaphore. Initialize the semaphore to N-1 (one less than the number of philosophers), wait for that semaphore before getting either chopstick, and signal it after releasing both chopsticks.
2. Allow a philosopher to only pick the chopsticks if both are available. This must be done in a critical section – i.e., testing and acquiring the chopsticks must be protected by a mutual exclusion semaphore.
3. Use an asymmetric solution – odd philosophers pick up their chopsticks left followed by right, while even philosophers pick up their chopsticks right then left. (This requires that each philosopher is assigned a unique id number – see the demonstration program written in class.)

Assignment

Implement one solution to the Dining Philosophers Problem for \( n \) philosophers.

- Each philosopher is represented by a Win32 API thread.
- Write your solution so that the number of philosophers is easily changed. A #define works well.
- Modify the partial solution of Figure 6.15
  - Choose and incorporate one of the text’s three “possible remedies”
  - Replace the do-loop with one that loops for approximately 60 seconds (demonstrated in class)

Your program must maintain the following descriptive information for each philosopher and print it out at the end of the program run. This is most easily done with two global arrays indexed by a philosopher’s id number.

1. The total number of times that the philosopher ate (increment a counter whenever the philosopher gets both chopsticks)
2. The total amount of time that the philosopher waits for the chopsticks (i.e., the elapsed time should not include either thinking time or eating time). The following pseudocode illustrates the partial solution:

   ```
   loop for about 60 seconds
   start = current time
   wait(left chopstick)
   wait(right chopstick)
   end = current time
   elapsed time += (end - start)
   eat -- sleep
   signal(left chopstick)
   signal(right chopstick)
   think -- sleep
   ```

   - Each philosopher should sleep for a random amount of time to simulate eating and sleep for a different random amount of time to simulate thinking. Sleep [0-4 seconds] for thinking and [0-2 seconds] for eating.
   - Run the program 4 times with an odd number of philosophers, about 1 minute duration each time.
   - Repeat this exercise but with an even number of Philosophers.
   - Create a simple report showing the statistics collected above (the number of times the philosopher ate, slept, and the total waiting time) and calculate averages for the even and odd runs.

Grading

- Upload your source code to Blackboard / WSU Online
- Upload a text file (or copy to the comments section), with the data collected and calculated during the 8 trial runs, to Blackboard / WSU Online