Read me file for Dining Philosopher's Problem

1. For Removing deadlocks without the use of synchronization primitives, I have implemented a strict ordering of resources, by which I mean that, philosopher of even number are first taking up their right fork first and philosopher of odd number are picking up their left fork first instead of all of the philosopher taking up their left first fock first. This resolves the deadlock. To implement this code, I have mead a lock technique which waits if the fock is not available.

2. For removing deadlocks with using synchronization primitives without worrying about ordering, I have used semaphores. Each fock is represented by a semaphore. There is also some kind of strict ordering of resources even in this one since normal semaphores results in deadlock.

3. Now to resolve the deadlocks without the use of synchronization primitives, now with only 2 bowls. So in dining philosopher's problem, if there is no deadlock, then only 2 philosopher can eat at a single time. And since there are two bowls, we give it to them. The key in this question is when to pick up the bowl. Bowl should only be picked up after a philosopher has picked up both the forks, not before them and put them down just after eating.It follows the same order of restriction that we implemented above.

4. Now to resolve the deadlocks with the use of synchronization primitives, now with only 2 bowls. So in dining philosopher's problem, if there is no deadlock, then only 2 philosopher can eat at a single time. And since there are two bowls, we give it to them. The key in this question is when to pick up the bowl. Bowl should only be picked up after a philosopher has picked up both the forks, not before them and put them down just after eating.To represent forks and bowl we have used semaphores and the rest is same as we have implemented above.