OPERATING SYSTEMS 2
PROGRAMMING ASSIGNMENT 5
Implement Dining Philosopher's using Conditional Statements

Implementation :

Philosopher's threads are created at the start of the program. The state of each thread is set to THINKING at the start. Writer function is called by each and every philosopher where they eat and think h times each.

There are three states of philosopher : **THINKING, HUNGRY and EATING**.

Reqtime notes the time at which a philosopher request for eating.
Entertime notes the time at which a philosopher enters the critical section.
Exittime notes the time at which a philosopher exits the critical section.

wait_time_w , worst_time_w - variables to calculate average waiting time and worst case waiting time for a philosopher.

At the start of thread, start time is saved in variable. After finishing current iteration, end time is noted and the difference is added to corresponding time variable, which is used to calculate average waiting over the iterations.

sleep function is called over to signify that the philosopher is thinking or eating at a particular time.

Design Of Program :

The program has been designed to solve the **Dining Philosopher Problem.**
The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the

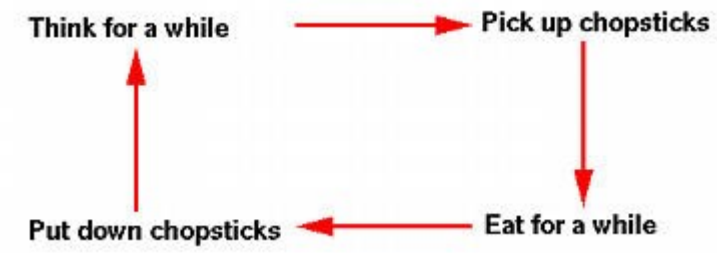
two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

<pthread.h> - for thread creation
<random> - To create random number generating engines.
<unistd.h> - To calculate current system time.
<fstream> - Creating file streams.
<mutex> - for creating locks
<chrono>
<ctime>
<condition_variable>- for implementing conditional statements.

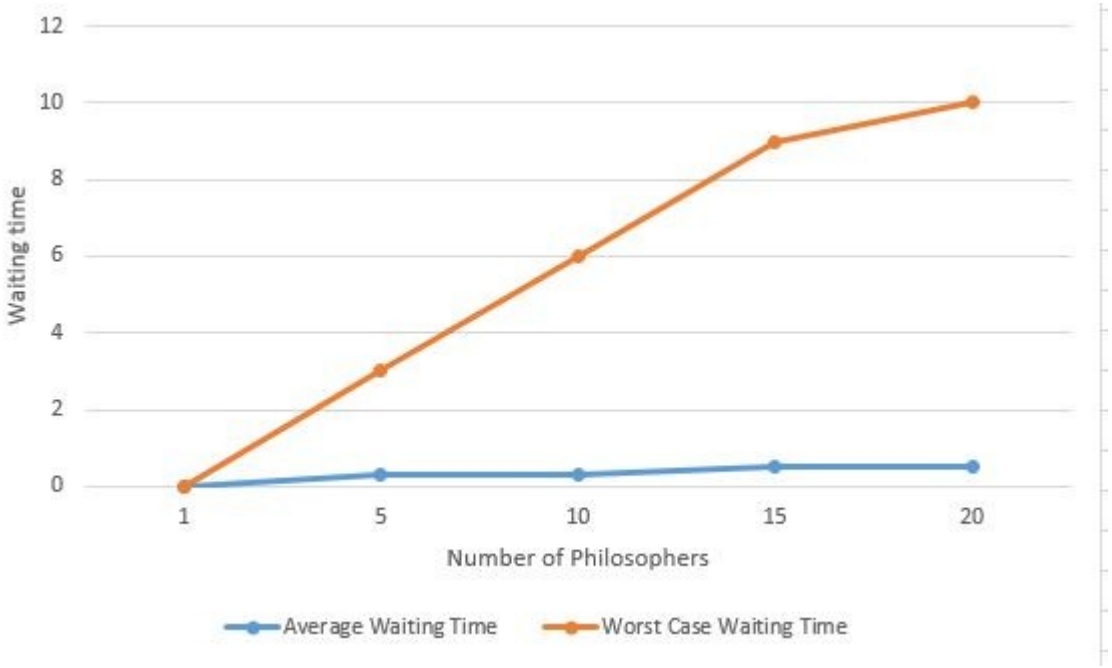
The problem was designed to illustrate the challenges of avoiding deadlock

Resource starvation might also occur independently of deadlock if a particular philosopher is unable to acquire both forks because of a timing problem. For example, there might be a rule that the philosophers put down a fork after waiting ten minutes for the other fork to become available and wait a further ten minutes before making their next attempt. This scheme eliminates the possibility of deadlock (the system can always advance to a different state) but still suffers from the problem of livelock.

Pickup() and **putdown()** functions are made such that no philosopher starves (i.e. wants to eat, but never gets a chopstick), and such that deadlock doesn't occur (a subset of the above, because it would mean that all philosophers starve...) It also attempt to try to minimize the amount of time that the philosopher's spend waiting for chopsticks.



GRAPH:



The graph shows that on increasing the number of philosophers thread , both Average waiting time and Worst Case waiting time increases.