Operating Systems-2 Assignment 5 Report

<u>Related Terms</u>:

- Entry Section: A segment of the code which is common to all the threads and where threads request & wait to enter the critical section.

- Critical Section(CS): A segment of the code where race conditions can happen. So, only one thread should enter this section at a time.

- Exit Section: A segment of the code where threads exit the critical section and another thread is allowed to enter the critical section.

- Remainder Section: The remaining segment of the code which is not critical i.e. where race condition will not happen.

- Waiting Time: It is the time that a thread waits in the entry section to get into the critical section.

Design & Implementation of code:

The design of the dining Philosophers is as follows:

- At the start of the program, main() reads input from “inp-params.txt” and store the values which are needed in the all threads globally.
- Then, we create an array of threads to execute the function dphil();
- We allocate memory for an array of type “struct phil” which contain the status and conditional variable for a philosopher.
- Function “test()” checks for a given philosopher if he/she can pick up the chopsticks to its left and right. If he/she can then it changes its status to eating and signal the corresponding conditional variable.
- Function “Pickup()” changes the status to hungry and calls “test()”. After execution of “test()” if a philosopher is not eating then it waits on its conditional variable.

- Function “Putdown()” changes the status of a philosopher back to thinking and then calls “test()” on its two immediate neighbours.
- To ensure atomic access to “Pickup()” and “Putdown()”, we use pthread_mutex_lock.
- We pass the “Phil” threads to its function using “pthread_create()” and all of the threads runs simultaneously.
- To generate random numbers, we use default_random_engine.
- To implement the exponential distribution of time, we use exponential_distribution<> in library <random>.
- Phil threads measures the time at which they request for the allocation of CS, they enter the CS and the time at which they exit
  from the CS by using functions from library <chrono> and prints that to a log file.
- To get the waiting time of each thread, we just subtract the timestamp at which it enters the CS and the timestamp at which it requests for the CS.
- To simulate the actions of eating and thinking, we sleep the thread. To implement it, we used “sleep()” which takes parameter in seconds.

- At the start of the Critical section, we call “Pickup()” and at the end of it, we call “Putdown()”.

<u>Graphs</u>:

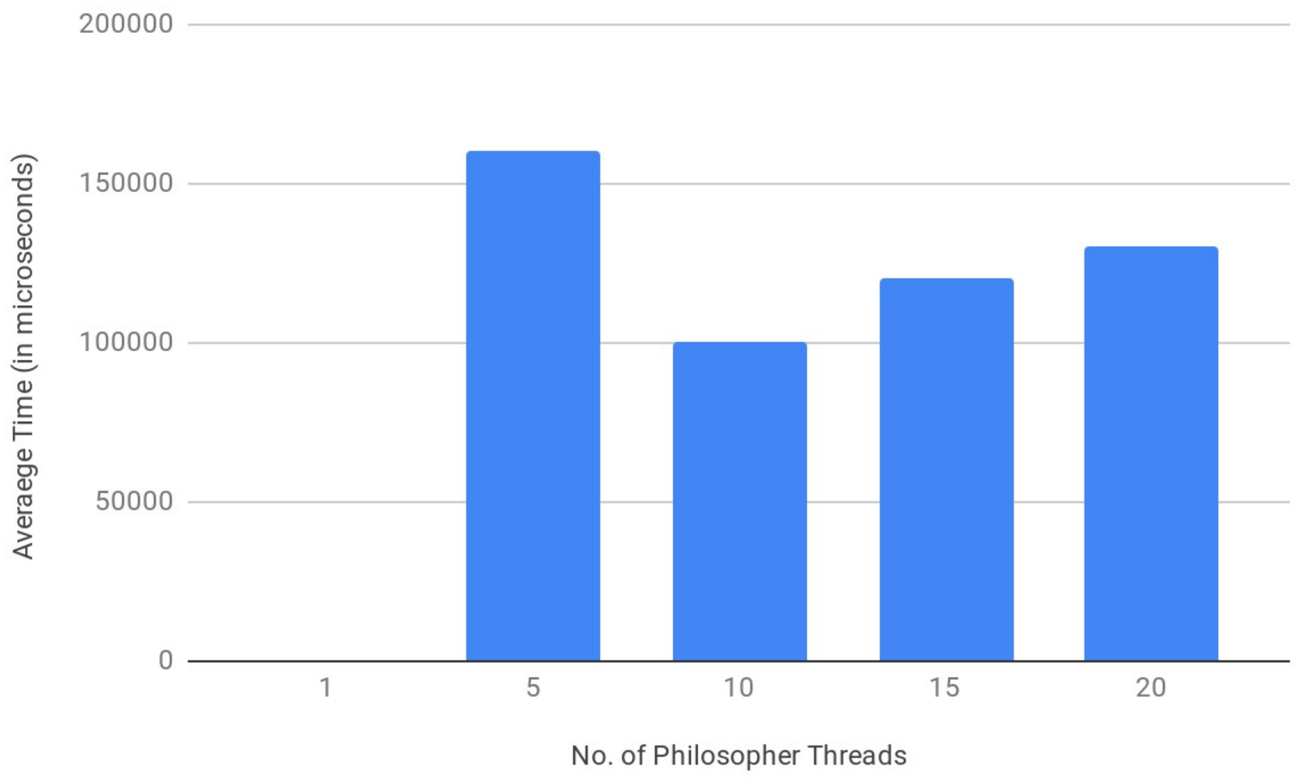
Graphs are plotted for

(h, µCS, µRem) = (10, 1, 2).

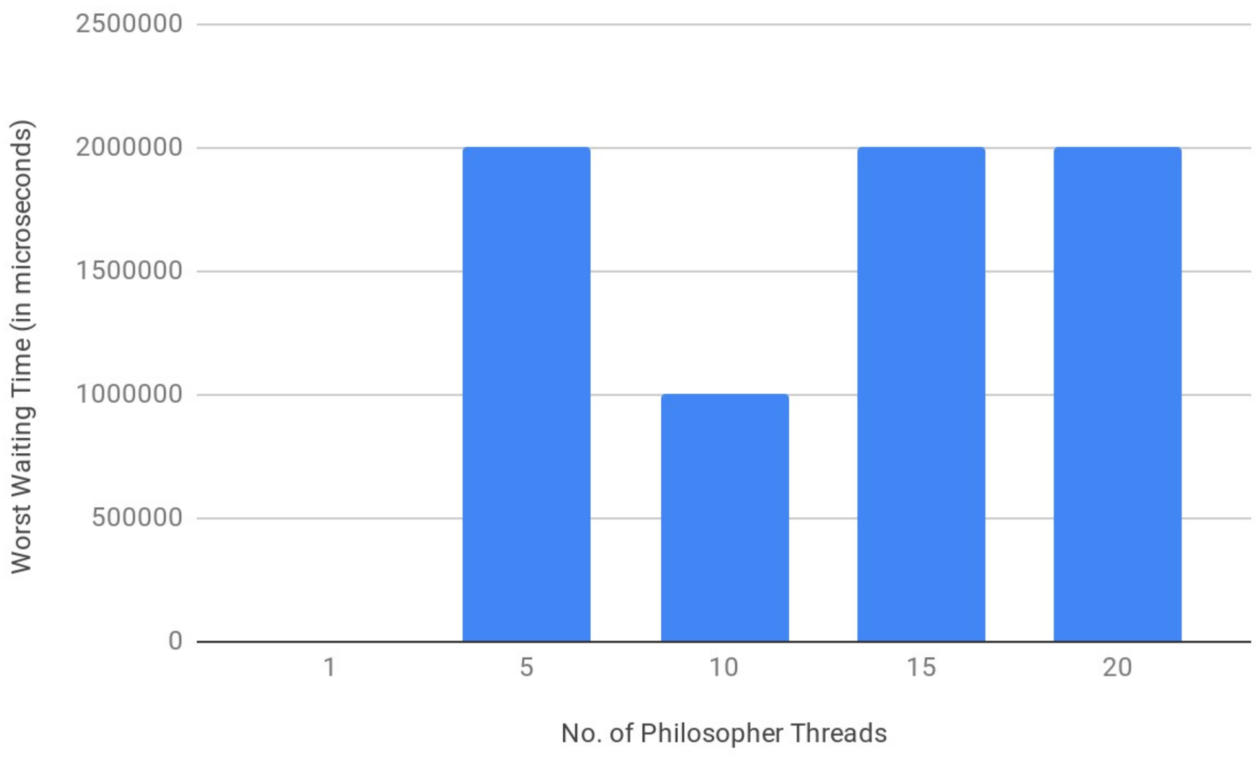
The value of n is varied from 1 to 20 in the increments of 5.

The graph for average waiting time and worst waiting time are plotted for the Philosopher threads.

Average Waiting Time :



Worst Waiting Time:



<u>Conclusion</u>:

As we can see from the graph, the average and worst waiting time for 1 thread is almost zero as it is going to enter the CS again and again. Average waiting time is increasing for increasing no. of threads. Worst waiting is almost similar for all no. of threads. For n = 5, Average waiting time is unexpectedly high.