Operating Systems 2
Name :- Ajinkya Bokade
Roll No :- CS17BTECH11001

Aim

To Implement Dining Philosopher's problem using Conditional Variables.

Design of Program

The input parameters to the program will be a file, named inp-params.txt, consisting of all the parameters:- n: the number of philosopher threads, h: the number of times each philosopher thread tries to enter the CS, µeat, µthink (delay values for CS and Remainder section that are exponentially distributed with an average of µeat, µthink milli-seconds.)

We are using pthread conditional variables and pthread mutex lock.
Pthread conditional variables support two operations:-
1)int pthread_cond_wait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex) :- This function shall block on a condition variable. They shall be called with mutex locked by the calling thread or undefined behavior results.
These functions atomically release mutex and cause the calling thread to block on the condition variable cond;

2) int pthread_cond_signal(pthread_cond_t *cond) :- This function shall unblock at least one of the threads that are blocked on the specified condition variable cond (if any threads are blocked on cond).

Implementation of Program

The n number of philosopher threads are created using pthreads. Each of the n philosopher thread is made to enter CS h times.
We have used n conditional variables for each thread.
Also, enum states is defined for each of the n philosopher threads which can have values "THINKING", "HUNGRY", "EATING".

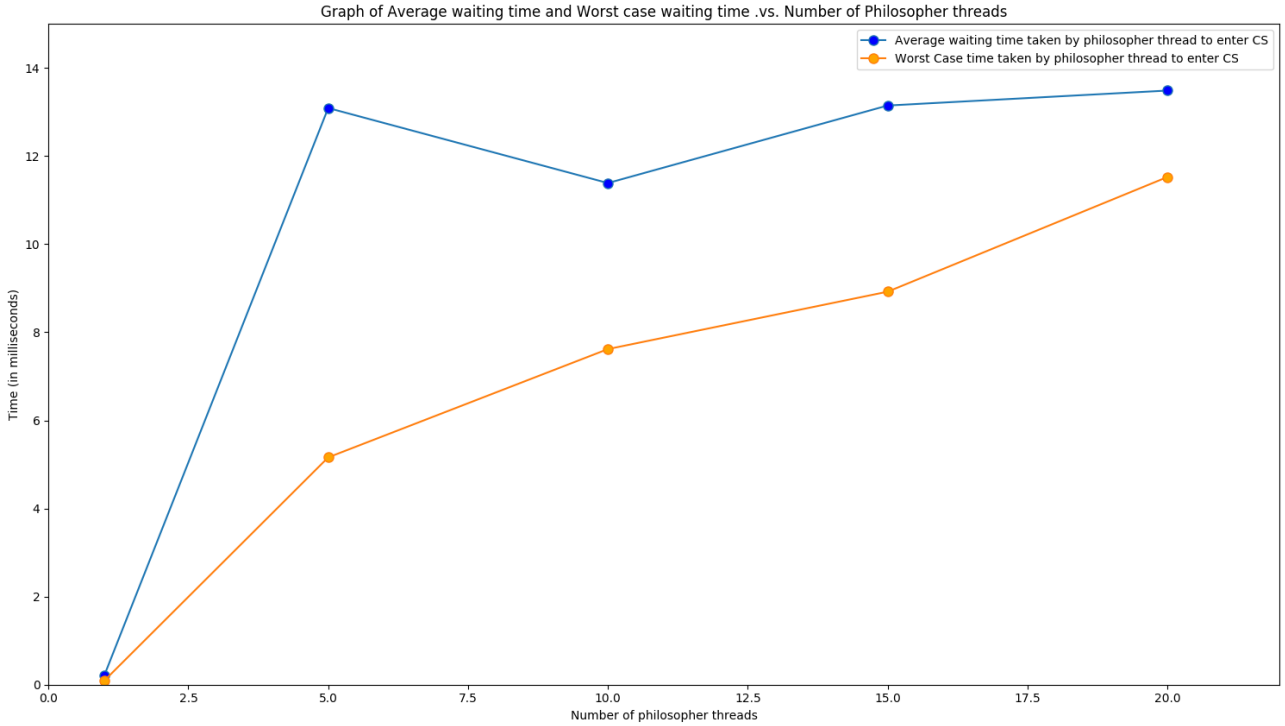
Three functions namely test, pickUp and putDown have been defined.
Test function takes an int parameter and checks if that philosopher thread can acquire both chopsticks (i.e. if the states of its neighbours are not EATING). If so, it changes that thread's state to EATING and signals conditional variable of that particular thread.
PickUp function takes an int parameter and changes the state of that thread to HUNGRY and calls the test function. If after calling, if its state doesn't become EATING, then it calls wait on that particular conditional variable of that thread. Also, before thread calls pickup function it has to acquire mutex lock which ensures mutual exclusion between threads.
PutDown function takes an int parameter and changes the state of the thread to THINKING and calls test function for its neighbouring threads. Here also, before thread calls putDown function it has to acquire mutex lock which ensures mutual exclusion between threads.
This solution is deadlock free but starvation may happen in this solution.

Comparison of the performance of producer and consumer threads

Input parameters :-
Count of each philosopher thread (h) = 10
μeat = 1
μthink = 2
These parameters are kept constant throughout.
n (Number of philosopher threads) have been changed from 1 to 20 in increments of 5.

Graphs

1) Graph of Average waiting time and Worst Case waiting time of Philosopher threads vs Number of philosopher threads (Count of each philosopher threads kept constant at 10)



Conclusion

We observe that average waiting time taken by philosopher threads increase slightly as number of philosopher threads increases and afterwards almost remains constant (varies very little). Although when number of philosopher threads were 5, it is somewhat higher. Also, worst case waiting time for philosopher threads increases slightly as number of philosopher threads increases which indicates the starvation of threads. Since as number of threads increase, it may happen that a philosopher thread may not get a chance to pickup both chopsticks for a longer amount of time which will lead to starvation.