Operating Systems-II
Report-Asgn-3

Task:

The goal of this assignment is to implement TAS, CAS, and Bounded Waiting with CAS mutual exclusion (ME) algorithms studied in the class. Implement these algorithms in C++.

Comparison metrics:

1. **Worst case waiting time:** the worst-case time taken by a thread to enter the CS in a simulation. This shows if threads are starving.

2. **Average Waiting Time:** The average time taken by a thread to enter the CS.

Approach and Implementation:

1. To achieve our goal I made a testCS function that takes thread_index as a parameter. We will pass this function along with the thread index to each thread and calculate the average waiting time and max waiting time for all three mutual exclusion algorithms.

   ```
   vector<thread> t; // Array of n threads
   for (int i = 0; i < n; i++)
          t.push_back(thread(testCS, i));
   ```

2. **`int usleep(useconds_t usec)`** function was used to suspend the execution of the thread for microsecond intervals and some other functions and data types from the Chrono library are used.

3. We make two exponential distributions and pass the value of λ1, λ2 in the constructor Later this can be used to obtain random numbers t1 and t2 with values that are exponentially distributed with an average of λ1, λ2 seconds. I used **template <class RealType = double> class exponential_distribution** this class to generate them.

4. The implementation of the critical section varies in ***testCS function*** with the change in mutual exclusion algorithm.

```
void testCS(int tid) {
 default_random_engine generator(time(NULL));
 exponential_distribution<double> d1(lambda_1); // critical section time
 exponential_distribution<double> d2(lambda_2); // remainder section time
 chrono::time_point<chrono::system_clock> start_time, end_time;

 for (int i = 0; i < k; i++) {
   // entry section
   fprintf(fout, "%d th CS Requested at %s by thread %d\n", i + 1,
           getSysTime().c_str(), tid + 1);
   start_time = chrono::system_clock::now();

   /*
       Mutual Exclusion algorithm implementation
       --Only this part changes
   */
   end_time = chrono::system_clock::now();

   // Critical Section Starts

   temp = (end_time - start_time);
   waiting_time += temp;                           // Calculating waiting time
   max_waiting_time = max(max_waiting_time, temp); // Updating max time taken
   fprintf(fout, "%d th CS Entered at %s by thread %d\n", i + 1,
           getSysTime().c_str(), tid + 1);
   usleep(d1(generator) * 1e3);
   fprintf(fout, "%d th CS Exited at %s by thread %d\n", i + 1,
           getSysTime().c_str(), tid + 1);

   lock.clear();

   // Critical Section Ends

   //Remainder Section Starts
   usleep(d2(generator) * 1e3);
   // Remainder Section Ends
 }
}
```

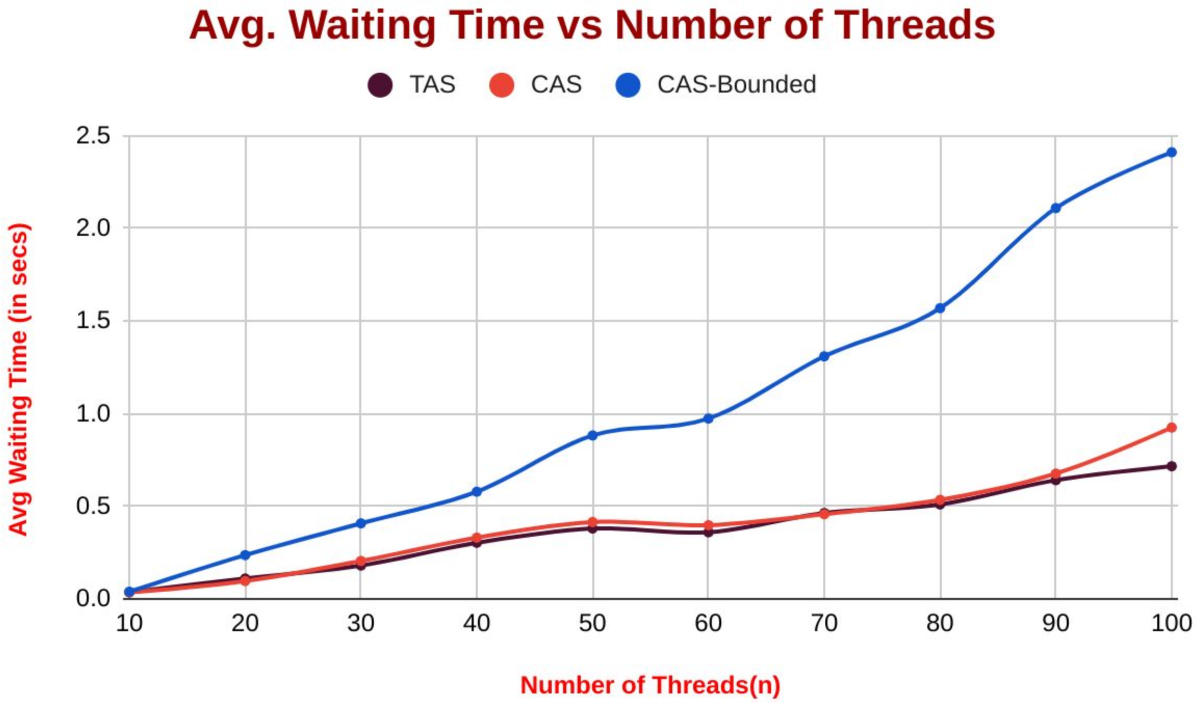
5. For Complete implementation details please refer to the code.
6. For printing logs, *fprintf/printf* are used instead of *cout* as cout streams causing mixing of logs.

Output Analysis:

(Plots are available at Google Sheets)

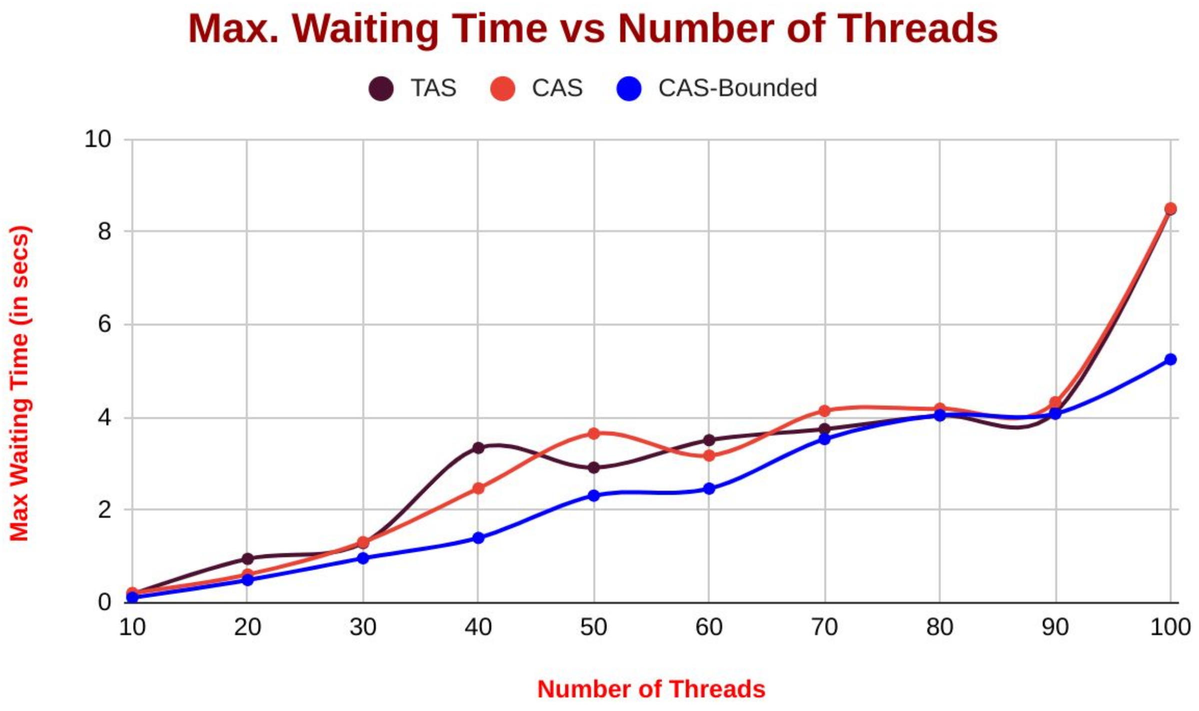
Input Params:

N: No. of threads varies from 10 to 100 in steps of 10.
K = 10: No. of CS request by each thread
λ1=20, λ2=30



Analysis:

1. Average waiting time taken by TAS and CAS algorithm is almost the same.
2. Average waiting time taken by Bounded-CAS is greater than both TAS and CAS as it ensures that no thread starve.



Analysis:

1. The worst-case waiting time taken by TAS and CAS algorithm is almost the same.
2. The worst-case waiting time taken by Bounded-CAS is much less than both TAS and CAS as it ensures that no thread starves.
3. The difference between the worst waiting times of Bounded-CAS and (TAS, CAS) increases as the no of threads increases.

Conclusion :

In terms of average waiting time, TAS performs best among all the three ME algorithms whereas CAS-bounded performs worst.

In terms of worst-case waiting time, CAS-bounded performs best among all ME algorithms whereas CAS/TAS performs worse.