Operating Systems 2

Tanmay Garg CS20BTECH11063

Programming Assignment 4

Program Design

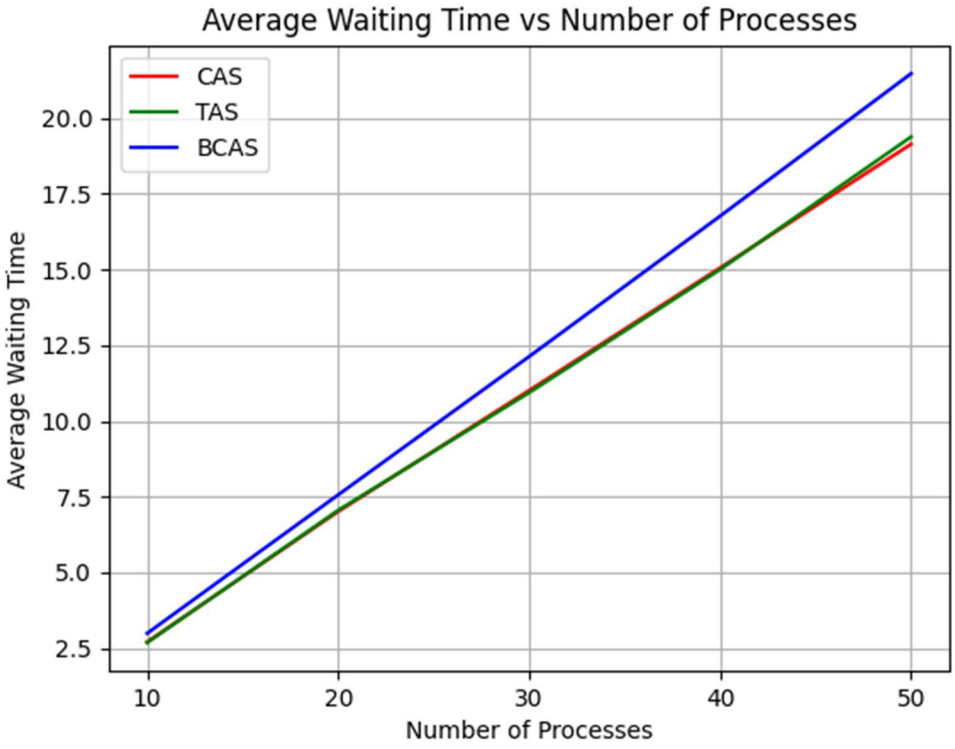
- The program take input from the file "inp_params.txt" and reads the Number of Processes, Number of Repeats and Values of λ_1, λ_2 for exponential distribution
- The program then creates n threads, and assigns parameters to these threads and calls the function *testCS()*
- The parameters are initialized in *thread_parameters_t* class with number of repeats, ID, values of λ_1, λ_2
- testCS() First copies the parameters into variables
- It then gets the system time for requesting the thread
- It then runs the lock part of the algorithms TAS (Test and Swap), CAS (Compare and Swap) and Bounded CAS mutual exclusion algorithms
- Once the lock is available, the thread locks it and gets the system time for when it enters to be run
- In order to simulate the critical section, the function uses *sleep()* and takes input of time using C++ library *std::default_random_generator* and *std::exponential_distribution*
- After sleeping it enters into the exit section where it gets the system time for when it exists
- It also clears the lock at this stage
- It then again sleeps to simulate remainder section same as before using uses *sleep()* which takes input of time using C++ library *std::default_random_generator* and *std::exponential_distribution*
- The entire process runs for *k* times as provided in the input file
- In the case of Bounded CAS, extra parameter is passed which is number of processes and the *testCS()* function is modified to incorporate Bounded CAS algorithm

Structure of Program

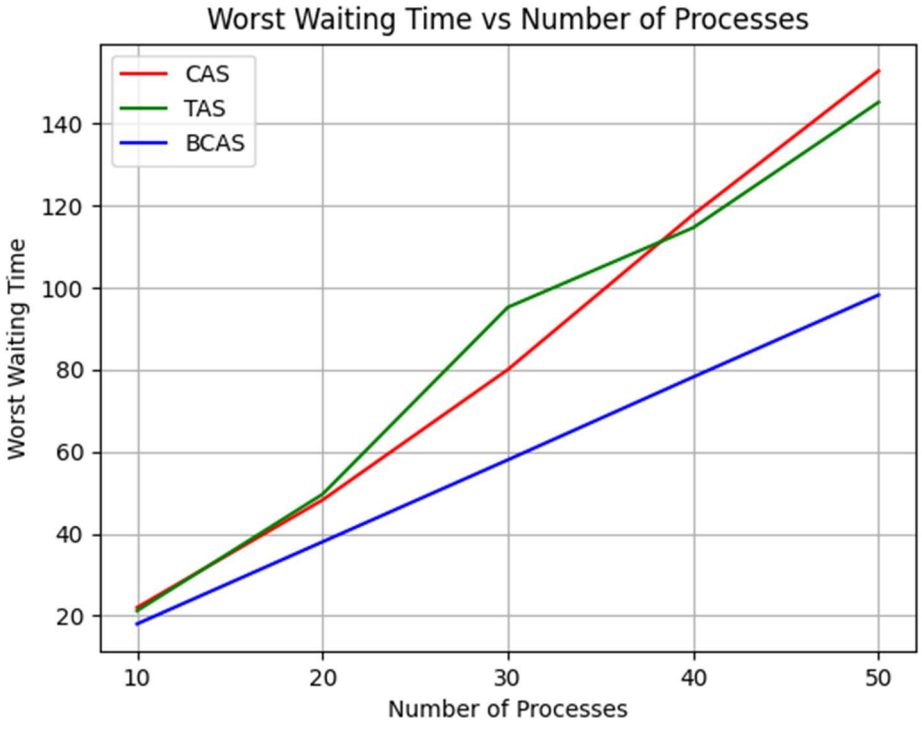
- We have a class
  - *Thread_parameters_t*
    - *ID*
    - *k (number of repeats)*
    - λ_1
    - λ_2
    - *n (number of processes – only in Bounded CAS)*
- *printTime_tm_ptr*
  - It takes *tm* pointer as argument and returns string of the time given as input
- *printtime*
  - It takes *time_t* as input and returns string of time given as input
- Global Variables
  - *Time_t GlobalWaitTime*: To store the Total wait time for all processes
  - *Time_t WorstWaitTime*: To store the worst wait time amongst all processes
  - *FILE* output_file_ptr*: Pointer to file where logs will be printed

        - FILE has been taken to allow faster I/O operations than using ofstream and instream
    - *std::atomic<int> lock_guard_*: Global lock variable
        - In the case of TAS this variable is: *std::atomic_flag*
- *testCS()*
    - This is the function passed to threads where the entire algorithm takes place for CAS, TAS and Bounded CAS

Comparison between Number of Processes and Average Waiting Time



Comparison between Number of Processes and Average Waiting Time



Note:

- The processes have been generated at random using *std::default_random_generator* and *std::exponential_distribution*
- There can be situations where there might be anomalies while running multiple threads and running them multiple times
- Compiler optimizations and system architecture and the operating system this program runs on might give some slightly altered results
- The results also depend on the values of λ_1, λ_2
- All the outputs generated in the stats file were copied and pasted into a python file to generate a graph

Conclusion

- The value of *n* goes from 10 to 50
- The value of *k* is fixed at 10
- The value of λ_1 is taken as 1
- The value of λ_2 is taken as 2
- All of them have been averaged by running 5 times
- In the case of Bounded CAS, it performs worse in average waiting time as it does not result in starvation and each thread is given equal chance to enter the Critical Section
- Whereas TAS and CAS both perform almost similarly
- In the case of Worst Waiting Time, Bounded CAS has least of the worst waiting times as it tries that no thread undergoes starvation
- Whereas TAS and CAS both perform almost similarly in the case of worst waiting time but worse than Bounded CAS due to starvation
- In both, the waiting times increase with the increase in *n*