Operating Systems 2

Tanmay Garg CS20BTECH11063

Programming Assignment 5

Program Design

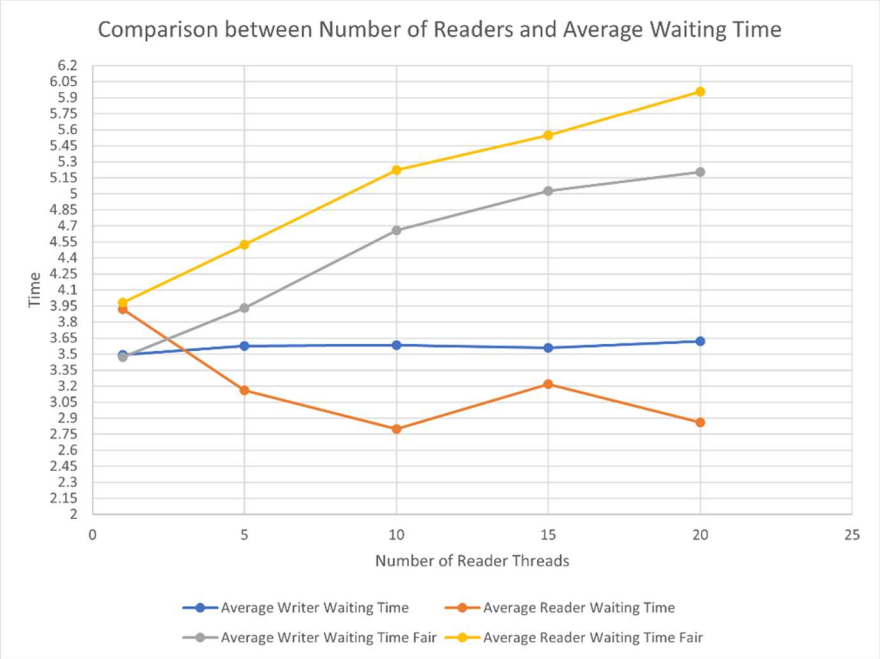
- The program take input from the file "inp_params.txt" and reads the Number of Reader Threads, Writer Threads, Number of times Reader and Writer threads need to repeat and values of μ_1, μ_2 for exponential distribution
- The program then creates the required number of reader and writer threads and calls the functions reader() and writer()
- The parameters are initialized in *thread_parameters_t* class with number of readers, writer threads, number of repeats of reader and writer threads, thread ID and μ_1, μ_2
- Both the reader() and writer() copy the parameters into their respective variables
- Writer()
    - It gets the system time for requesting for the thread
    - It then waits for rw_mutex semaphore
    - It then gets the system time for entering
    - Sleep() is used to simulate the thread writing in CS
    - It then signals the rw_mutex semaphore and gets the exit time
    - Sleep() is again used to simulate the remainder section
- Reader()
    - It gets the system time for requesting the thread
    - It then waits for mutex
    - Based on the value of read_count it will wait for rw_mutex also
    - It then signals mutex and gets the entry time for the thread
    - Sleep() is used to simulate the thread writing in CS
    - It then waits for mutex and based on the value of read_count it signals rw_mutex
    - Mutex is then signaled and gets the exit time
    - Sleep() is again used to simulate the remainder section
- In the case of Fair Reader Writer, an extra in_mutex semaphore is used to avoid the problem of starvation of writer processes
- 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*
- 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*

Structure of Program

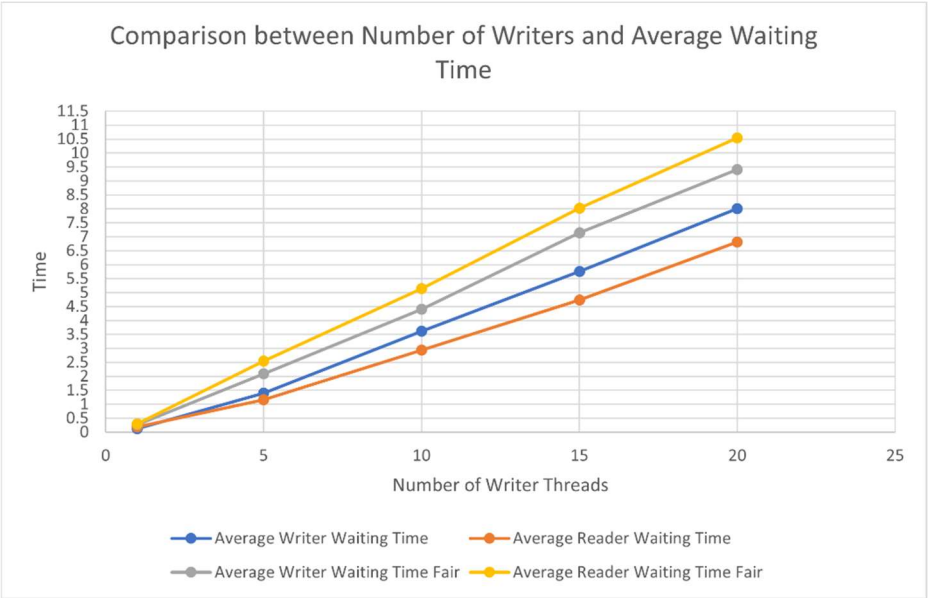
- We have a class
    - *Thread_parameters_t*
        - *ID*
        - *nw (number of writer threads)*
        - *nr (number of reader threads)*
        - *kw (number of repeats of writer thread)*
        - *kr (number of repeats of reader thread)*
        - $\boldsymbol{\mu_1}$

        - μ_2
- *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_Reader*: To store the Total wait time for reader processes
    - *Time_t GlobalWaitTime_Writer*: To store the Total wait time for writer processes
    - *Time_t WorstWaitTime_Reader*: To store the worst wait time amongst reader processes
    - *Time_t WorstWaitTime_Writer*: To store the worst wait time amongst writer processes
    - *FILE* output_log_ptr*: Pointer to file where logs will be printed
        - FILE has been taken to allow faster I/O operations than using ofstream and instream
    - *Int read_count: Used to count number of readers reading*
- *writer()*
    - This is the function passed to writer threads where the entire writer algorithm takes place
- *reader()*
    - This is the function passed to reader threads where the entire writer algorithm takes place

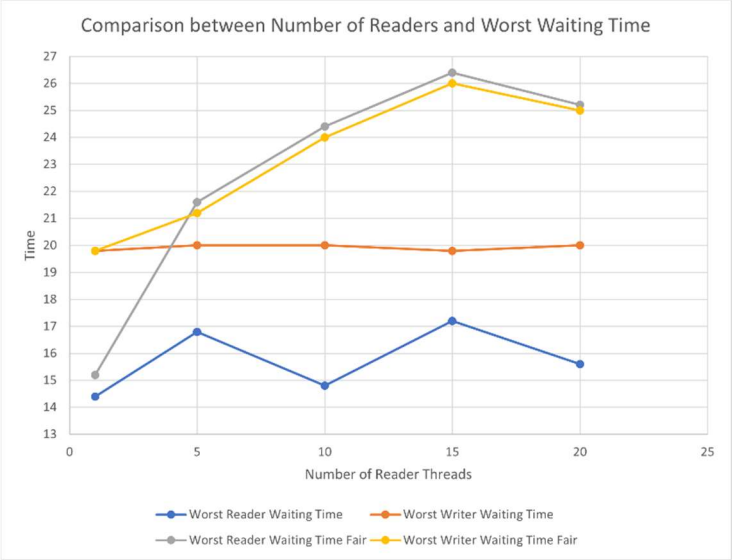
Comparison between Number of Readers and Average Waiting Time



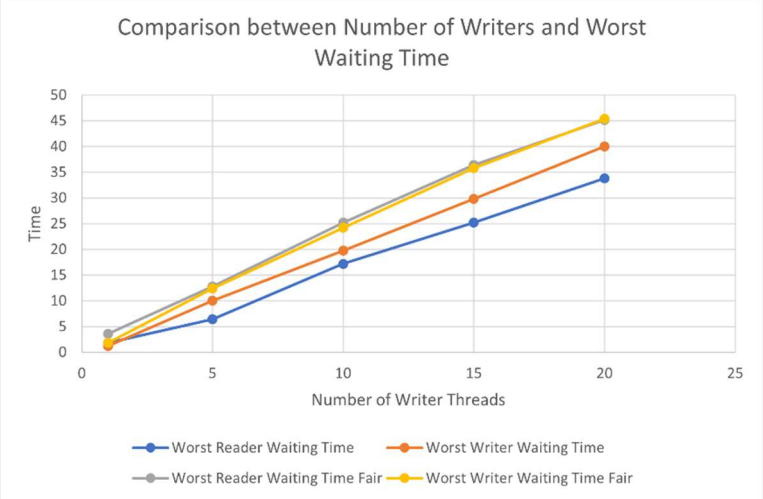
Comparison between Number of Writers and Average Waiting Time



Comparison between Number of Readers and Worst Waiting Time



Comparison between Number of Writers and Worst 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 μ_1, μ_2
- All the outputs generated in the stats file were copied and pasted into a excel to generate a graph
- The code for fair readers writers is taken from this research paper and was discussed in class

Conclusion

- The value of number of reader threads is taken as 1, 5, 10, 15, 20 where number of writer threads is kept constant at 10 and number of repeats for both reader and writer threads is kept at 10
- The value of number of writer threads is taken as 1, 5, 10, 15, 20 where number of reader threads is kept constant at 10 and number of repeats for both reader and writer threads is kept at 10
- The value of μ_1 is taken as 1
- The value of μ_2 is taken as 2
- All of them have been averaged by running 5 times
- In the case when number of writers are constant
    - Average Waiting Time of Readers in Fair RW is more than RW because in RW writer processes undergo starvation, but in Fair RW this starvation is avoided and the average waiting time for reader processes increases
    - Average Waiting Time of Writers in Fair RW is more than RW is also due to fairness between Readers and Writers in Fair RW
    - Worst Waiting Time both readers and writers are almost same in Fair RW which shows that none of the processes are starving
    - Worst Waiting Time for readers is more in Fair RW due to fairness to both readers and writers which in turn increases the waiting time for readers
- In the case when the number of readers is constant
    - Average Waiting time increases for all the processes as the number of writer processes increases
    - Average Waiting time for readers is more in Fair RW than in RW as in normal RW, the writer processes were starving but in Fair RW, starvation is prevented and average waiting time increases
    - Worst Case waiting time increases for all the processes as the number of writer processes increases
    - Worst case waiting time of reader and writer is almost same in Fair RW, this means the none of the processes are starving
    - Worst case waiting time for reader is more in Fair RW than in RW as the writer processes are not starving and the waiting time is increasing