Assignment 3 Report

Tanmay Goyal
AI20BTECH11021

Program Design:

The program begins with reading from the input file 'inp-params.txt' and getting the values of the number of threads, number of requests for the critical section, λ_1 and λ_2. If the file cannot be opened, we return an error.

We then define a class to store the parameters to be passed to the threads, which consist of the following attributes: The total number of threads, the number of critical section requests, the thread id, λ_1 and λ_2, and the pointer to the output file. This is because we will allow each thread to generate its own times for sleeping, and write to the output file by itself, instead of letting the main thread do it.

The basic design is as follows: we define a global variable which is atomic in nature. We also define two global variables to hold the average time for entering and the worst-case time for entering. For each algorithm, we define the entry section. Once the thread can enter the critical section, we update the output file, add the time taken for entering to average time, recompute the worst-case time for entering, make the thread sleep for a while to simulate some time-consuming job, and finally, let the output file know that the thread is exiting. We then define the exit and remainder sections. The remainder section is also simulated by making the thread sleep for a certain time. The time for sleeping is generated from an exponential distribution, with means λ_1 and λ_2 for the critical and remainder sections respectively. This entire procedure is repeated.

After all the threads are done, the main thread computes the average and worst case times, and prints it in the output file and on the terminal.

For Test and Swap, we simply define an atomic flag, and use the test_and_swap function.

For compare and swap, implementation becomes tricky since the *compare_exchange_strong* function defined in C++ is slightly different from the implementation studied in class.

The *compare_and_swap* function is defined as follows:

```
bool compare_and_swap(bool* flag , bool expected , bool new_val){
        bool temp = *flag;
        if(*flag == expected) *flag = new_val;
        return temp;
}
```

However, the *compare_exchange_swap* function is described as follows:

```
bool compare_exchange_strong(bool* flag , bool expected , bool new_val){
        if(*flag == expected) {
                *flag = new_val;
                return true;
        }
        else{
                expected = *flag;
                return false;
        }
}
```

Here, we thus, return if a successful swap was made or not, and not the original value of the flag. This resulted in some differences to the entry section of the code as compared to the textbook.

In the Compare_and_Swap program, we keep redefining *expected* inside a while loop, since *compare_exchange_strong* replaces *expected.*

In the compare_and_swap-bounded program, we define *key = false* so that every time a swap is not made, i.e. *flag != expected* we return to the loop.

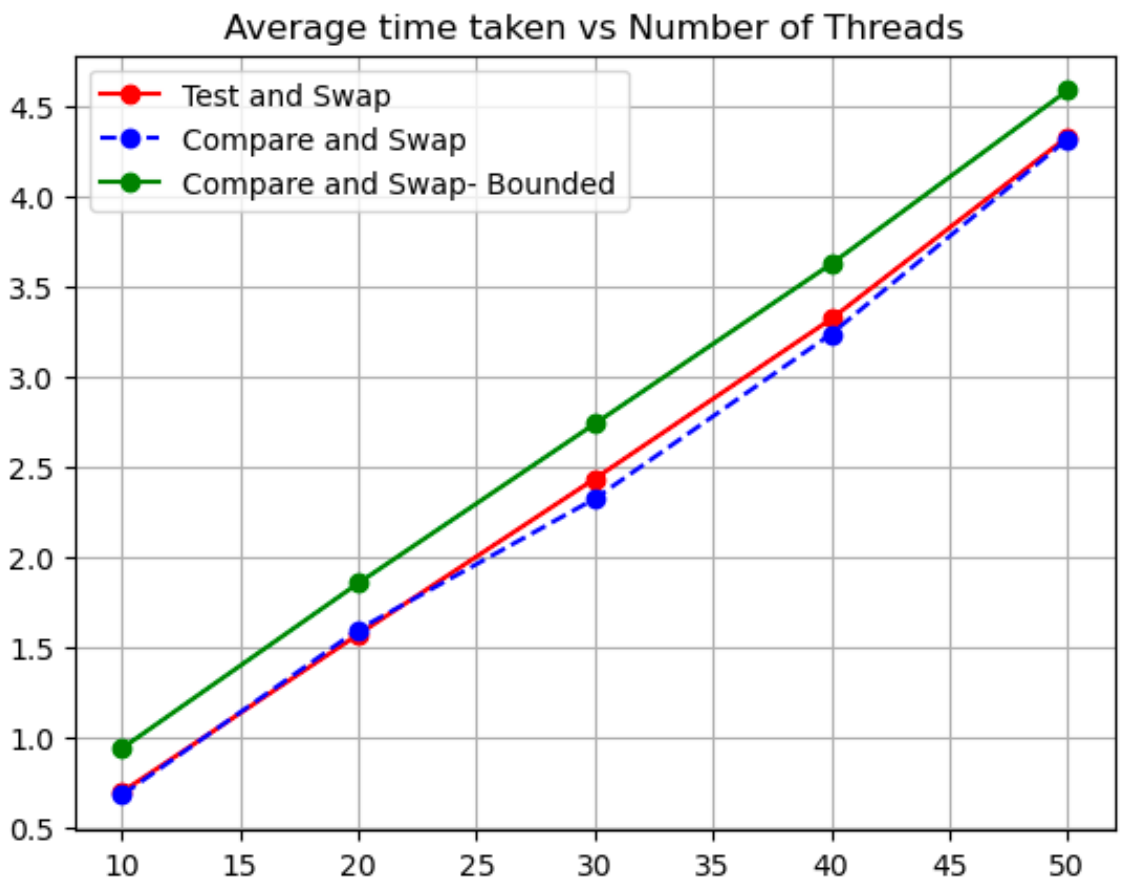
Analysis:

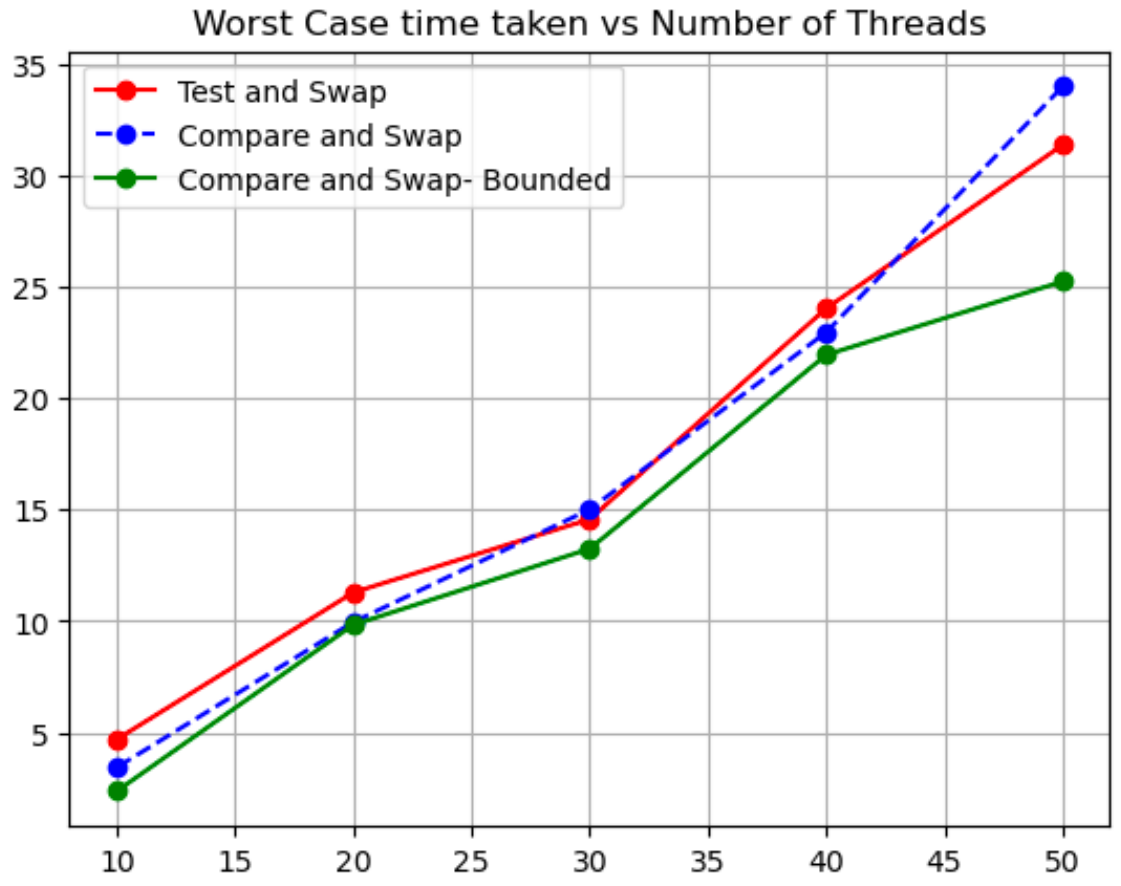
We keep the number of requests to the critical section fixed at 10, and $\lambda_1 = 10$ and $\lambda_2 = 15$

We find that the average time taken by the 3 algorithms is as follows:
Bounded-CAS > CAS ≈ TAS
We find that the worst-case time is as follows:
Bounded-CAS < CAS ≈ TAS

This shows that since the worst-case time for Bounded CAS is the least, there is very little starvation amongst the processes. However, what this means is the algorithm takes more time on average to choose which process enters the critical section.

CAS and TAS are almost similar in terms of their average times for entering critical section and worst-case times for entering critical sections, and there is lot of fluctuations.


Average time taken vs Number of Threads
Test and Swap
Compare and Swap
Compare and Swap- Bounded
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
10
15
20
25
30
35
40
45
50


Worst Case time taken vs Number of Threads
Test and Swap
Compare and Swap
Compare and Swap- Bounded
35
30
25
20
15
10
5
10
15
20
25
30
35
40
45
50