OPERATING SYSTEMS

ASSIGNMENT-II REPORT

GROUP 35

GROUP MEMBERS

Adarsh Kumar Rai	*f20191071@hyderabad.bits-pilani.ac.in*
Bhavya Tibrewala	*f20191404@hyderabad.bits-pilani.ac.in*
Evanston FX	*f20190909@hyderabad.bits-pilani.ac.in*
Kanishk Yadav	*f20191452@hyderabad.bits-pilani.ac.in*
Shantanu Kumar	*f20190375@hyderabad.bits-pilani.ac.in*

PROBLEM STATEMENT

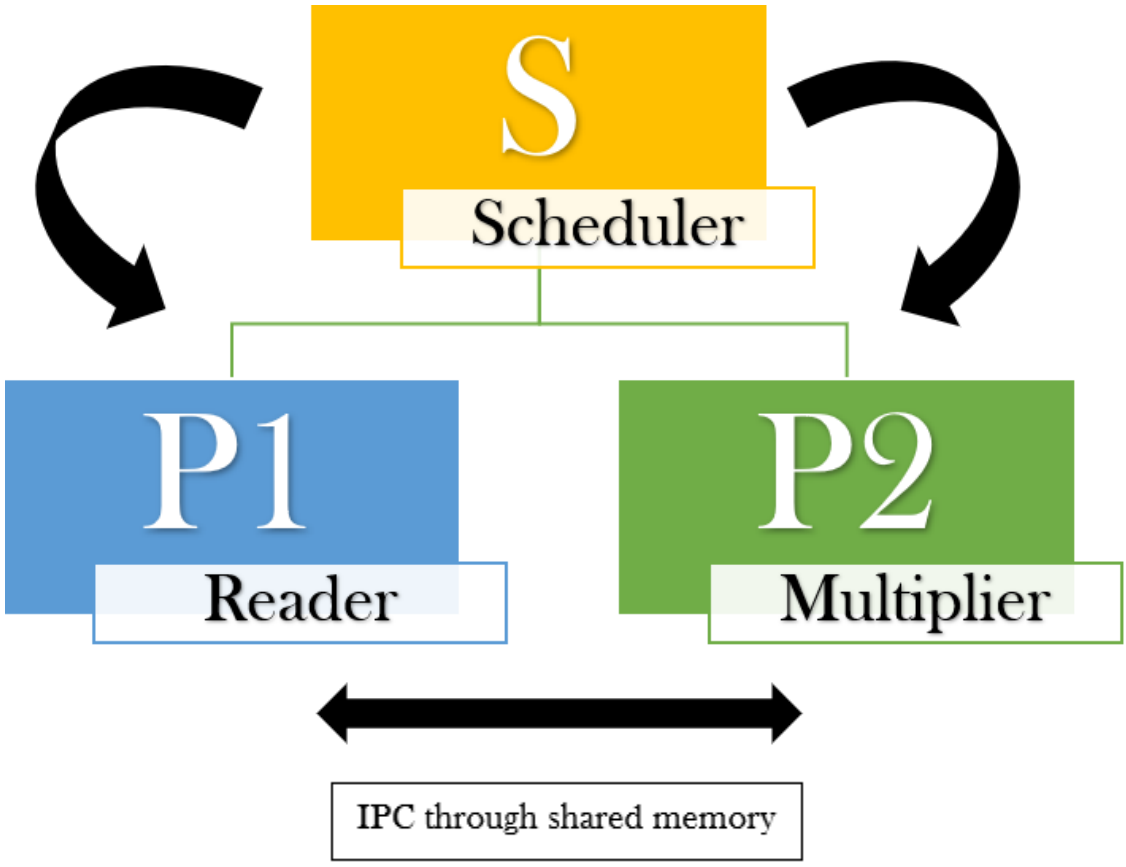
Multiply arbitrarily large matrices using parallelism provided by the Linux process and threads libraries. The scheduler program S spawns 2 children processes which exec to become the processes P1 and P2.

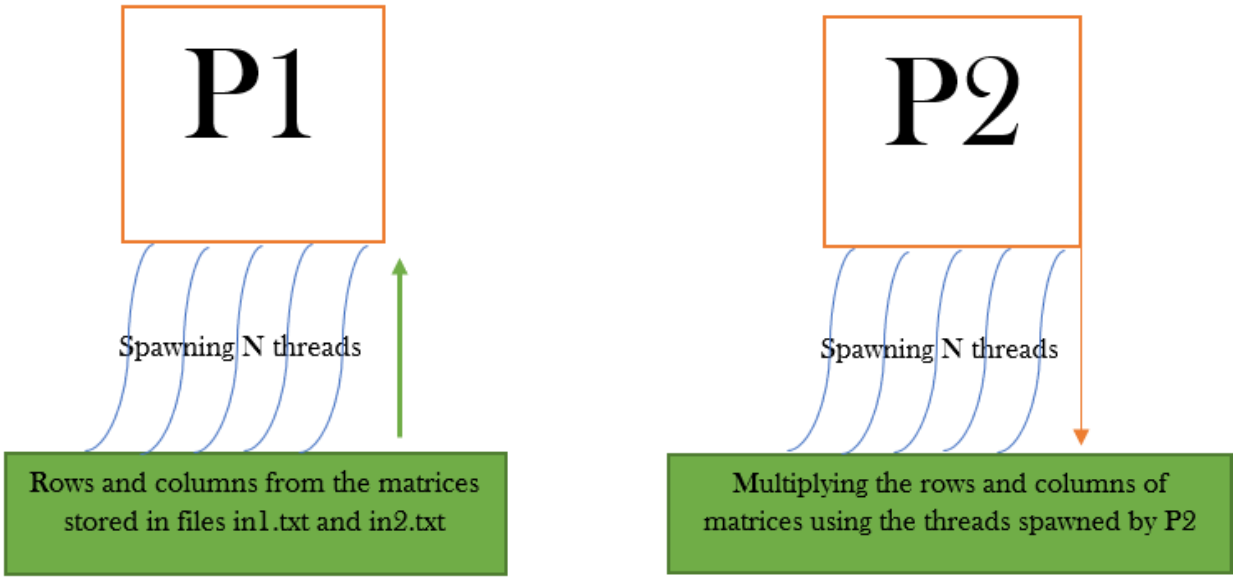
1. **P1** spawns n threads which each read row(s) and column(s) each from in1.txt and in2.txt.

2. **P2** uses IPC mechanisms to receive the rows and columns read by P1. P2 spawns multiple threads to compute the cells in the product matrix. The program stores the product matrix in the file out.txt

3. **S** uses some mechanism to simulate a uniprocessor scheduler. That is, it suspends P1 and lets P2 execute, and vice versa. Aim is to simulate the following scheduling algorithms in S:
   i) Round Robin with time quantum 2 ms
   ii) Round Robin with time quantum 1 ms

DIAGRAMMATIC REPRESENTATION OF THE PROBLEM





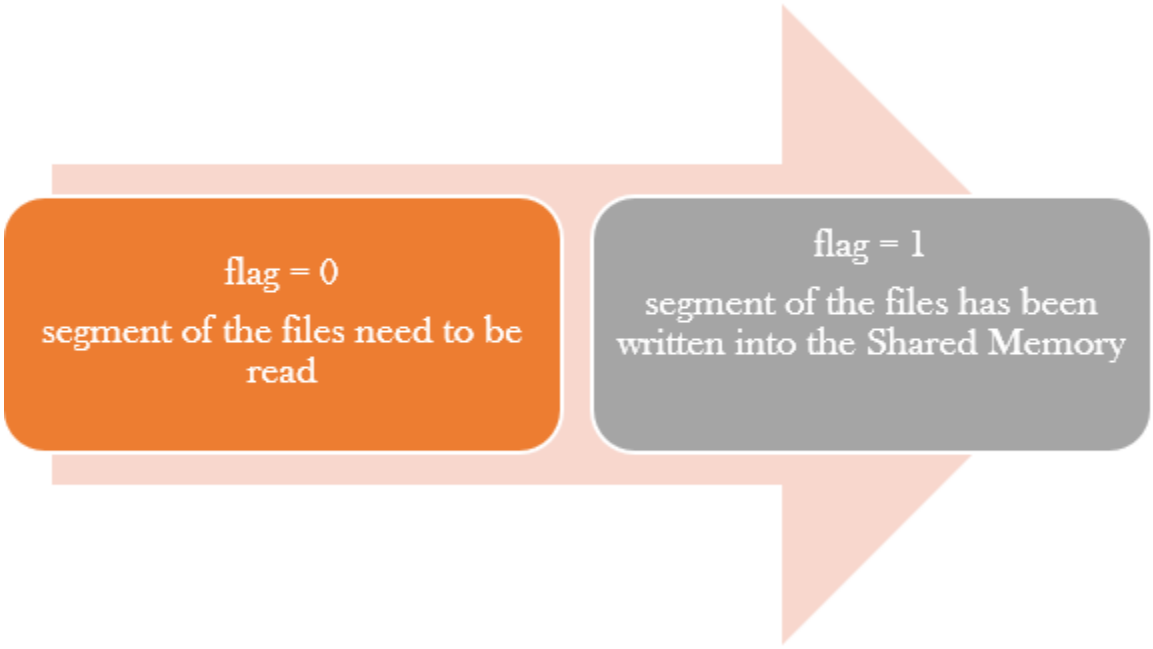
ROUND ROBIN SCHEDULING ALGORITHM

- Designed specifically for time-sharing systems and is similar to FCFS scheduling with preemption added for processes (P1 & P2) requiring time more than the time quantum (time slice) defined, helping switch between the processes (P1 & P2).
- The ready queue is treated as a circular queue.
- The ready queue is treated as a FIFO queue of processes with the new processes added at the tail of the queue.
- CPU picks the first process from the ready queue, sets a time (time quantum), and dispatches the process.

PROCESS P1: READER

1. The user inputs the number of threads and depending on the dimensions of the matrix, the matrix is divided into segments such that:
    - If the number of rows are divisible by the number of threads, then equal sized segments i.e. segments containing the same number of rows are distributed among the threads to be further written into the Shared Memory.
    - Else, if the number of rows are not divisible by the number of threads, then (n-1) threads are allotted equal sized segments and the nth remaining thread is allotted the remaining number of rows for the last segment.
2. Doing so, each thread is given a different part of the matrix to read and then write into the Shared Memory.

3. Since, each of the created threads are made to access different parts of the matrix, this holds the principle of mutual exclusion ensuring no race condition to occur during execution.
4. Depending on the number of threads, equal numbers of flags are created. For all threads, the flag is initially set to 0 and as soon as the thread reads a segment, the flag is set to 1.

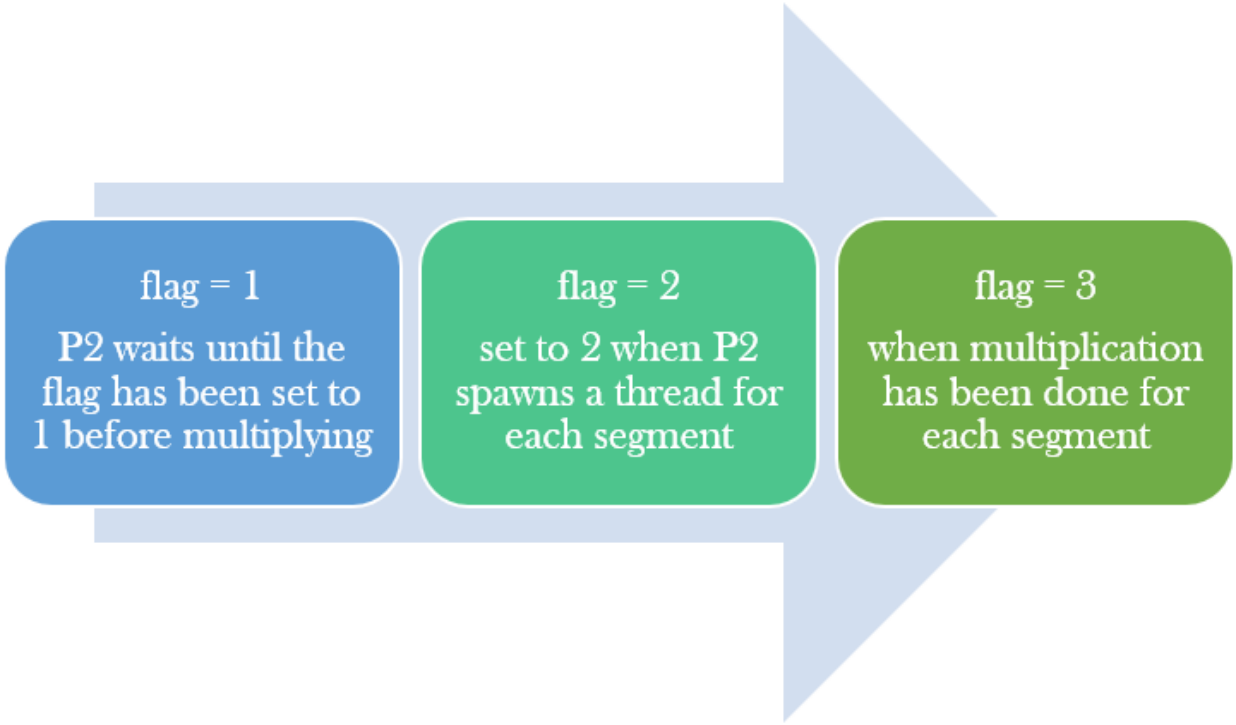


PROCESS P2: MULTIPLIER

1. The process P2 waits for the process P1 to read and send data from the input text files.
2. P2 checks whether the flag is set to 1 before spawning a new thread ensuring that its segment of the matrix is written into the shared memory before multiplying.
3. P2 spawns a new thread for each segment which was shared through the shared memory by P1 through Inter-Process Communication (IPC), setting the flag to 2 indicating that a

thread has been created for that particular segment as shown in the diagram.

4. The spawned thread multiplies its respective segment of matrix A with the matrix B and stores it in the matrix C. When the multiplication is completed, the flag for that particular thread is set to 3.
5. P2 waits for the all flags of all the threads to be set to 3 before writing the output matrix C into the *out.txt* file.

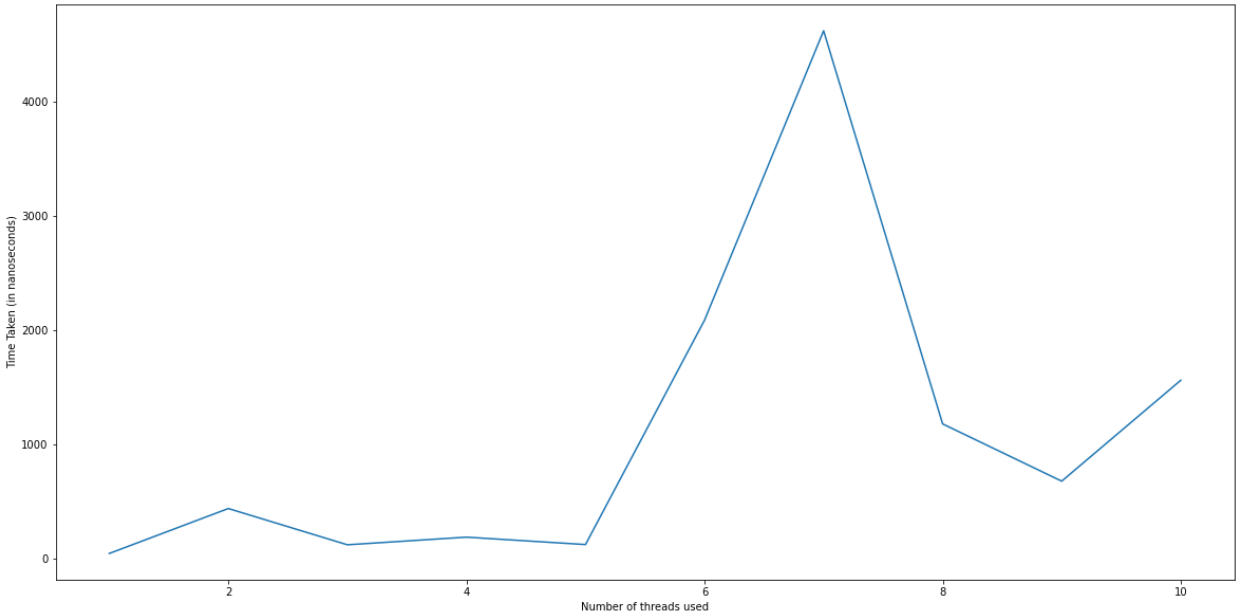


SCHEDULER

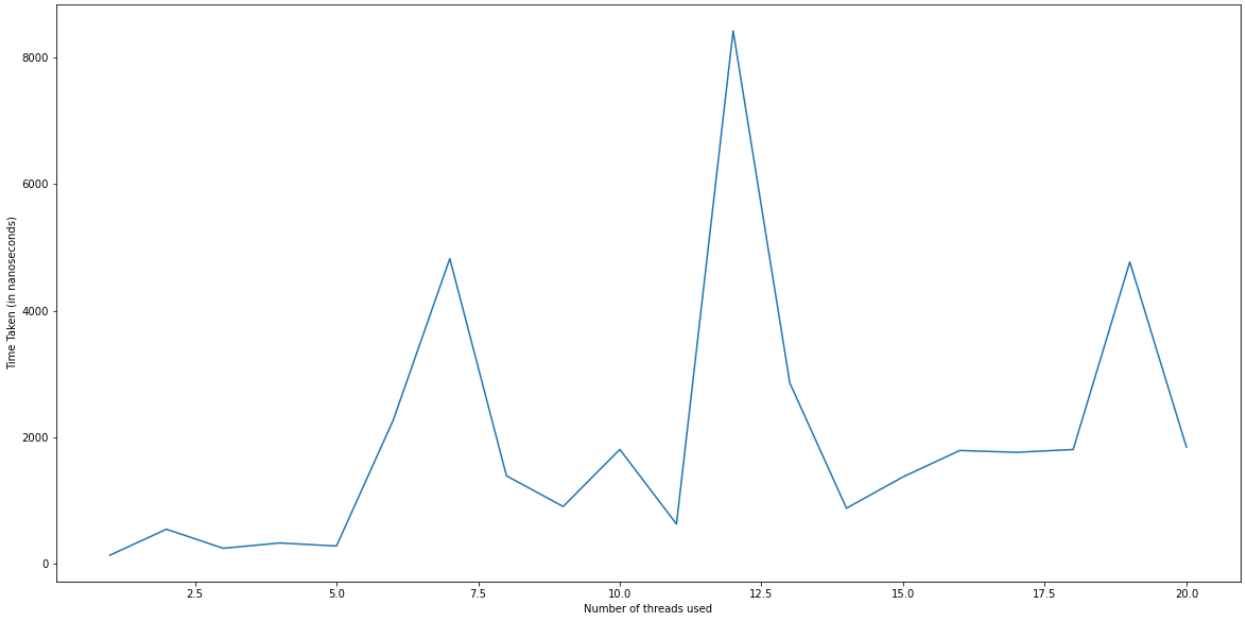
1. The execution of the scheduler is the key part of the entire implementation.
2. It spawns two processes P1 and P2, which operate as shown above using a set of flags for each thread created.
3. S suspends P1 and lets P2 execute, and vice versa employing a round robin approach which takes in time quantum as an input before execution and scheduling.

GRAPHICAL ANALYSIS FOR ROUND ROBIN SCHEDULING

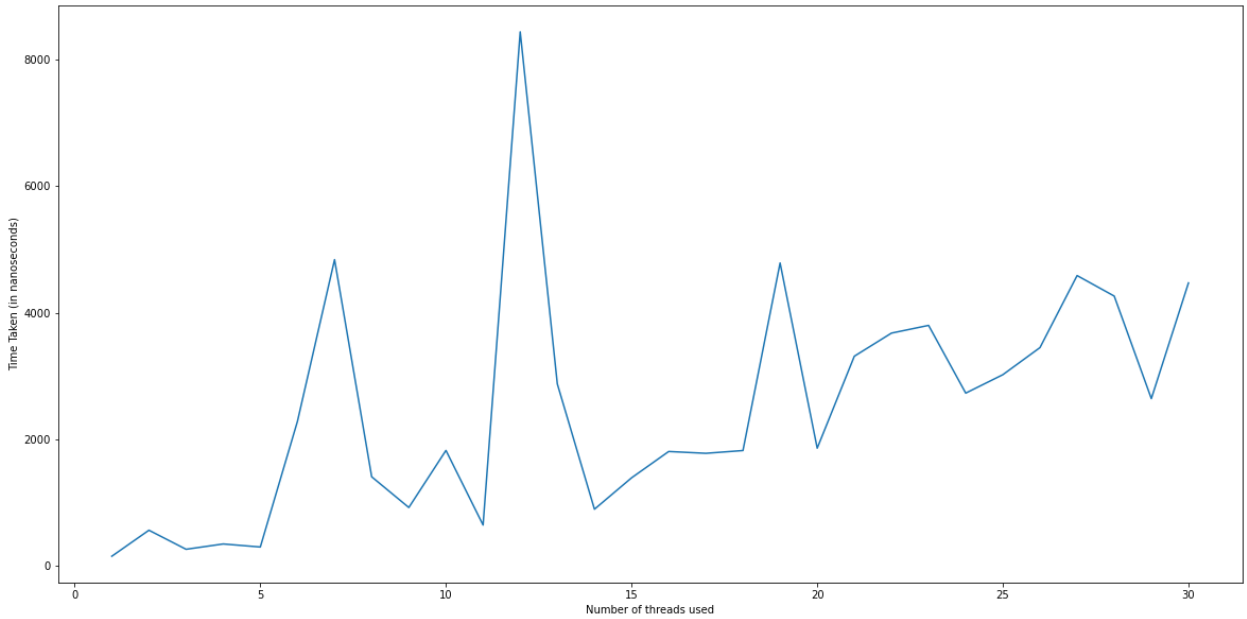
1. **Plot for the output multiplication of a (10x10) output matrix**



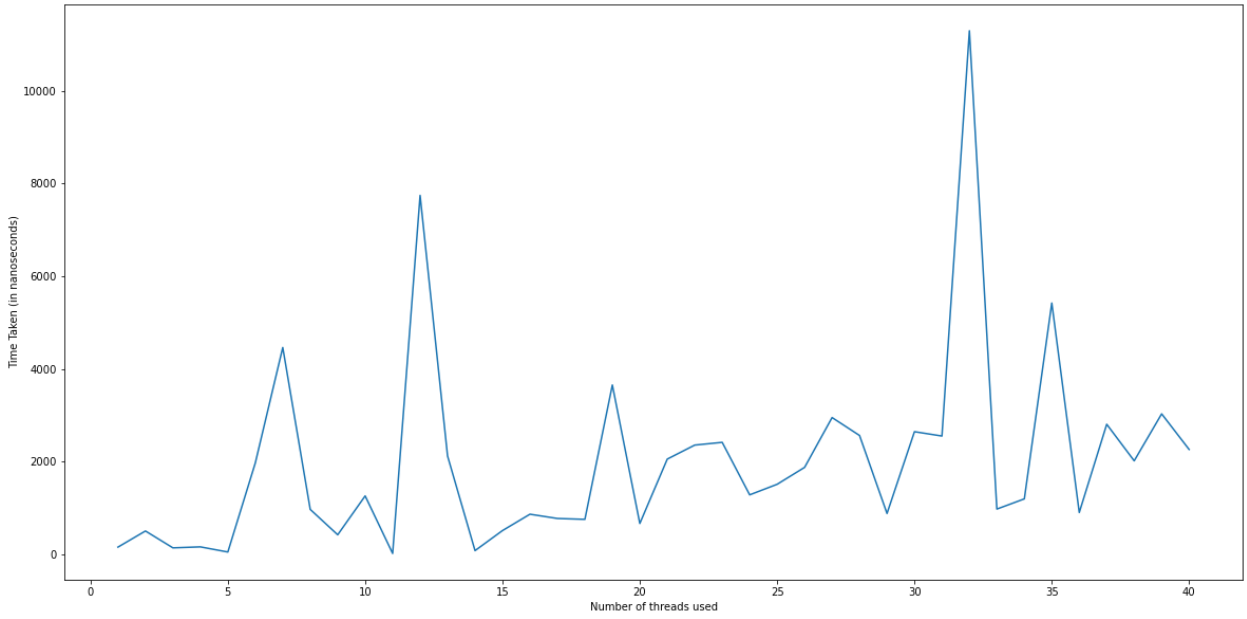
2. **Plot for the output multiplication of a (20x20) output matrix**



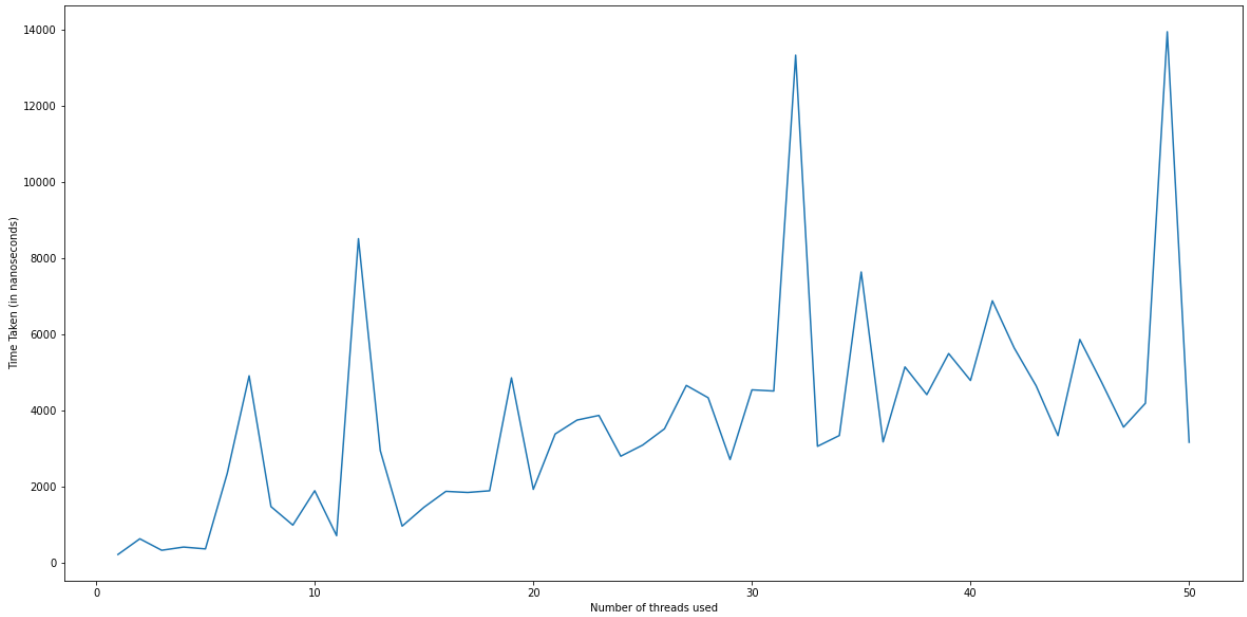
3. Plot for the output multiplication of a (30x30) output matrix



4. Plot for the output multiplication of a (40x40) output matrix



5. **Plot for the output multiplication of a (50x50) output matrix**



Analysis of the graphs produced from the execution of the scheduler

1. For every graph plotted above, we notice an outlier to be always present irrespective of the size of the output matrix.
2. Even though multithreading is implemented, the number of multiplication operations taking place isn't much compared to the cost of starting the thread; that's why a fairly good performance is observed with single threaded execution.
3. We also notice that unless working with extremely large matrices (with dimensions in the range of 1000), we are unlikely to see much improved results from a multithreaded approach.