OPERATING SYSTEM (CS F372)

Assignment II

By

Group 40

NAME	ID NUMBER
Abhisht Rustagi	2020A7PS1891H
Chetan Akash Vankadara	2020A7PS2196H
Kartik Kuckreja	2020A7PS1702H
Kunchala Sri Vatsav Reddy	2020A7PS0274H
Penugonda Satya Sohan	2020A7PS0190H
Varad Asawa	2020A7PS0217H

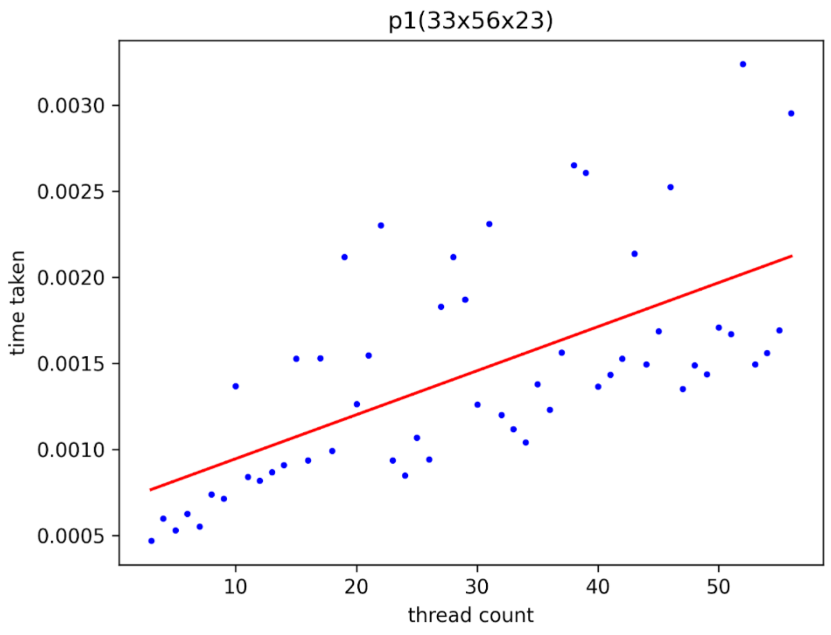
Table of Contents

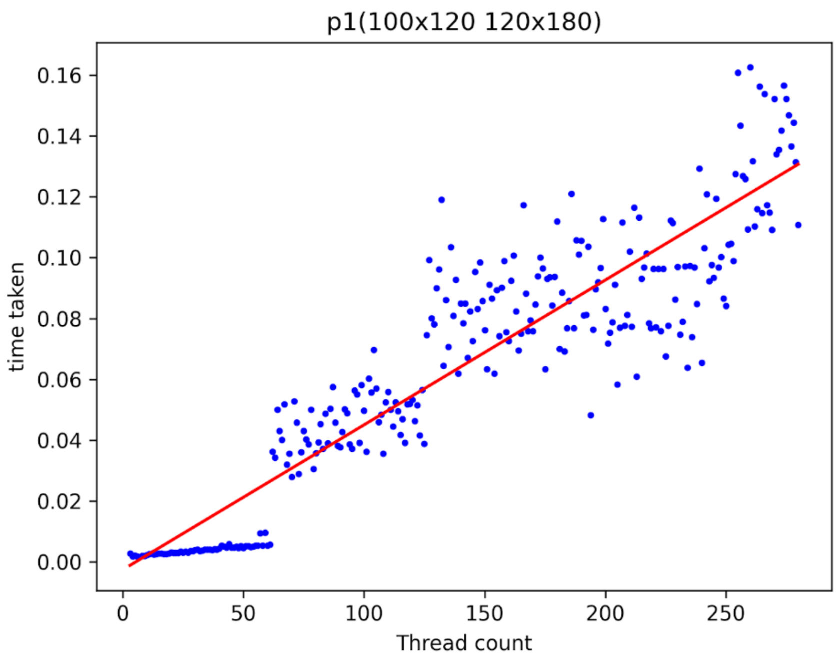
Overview:

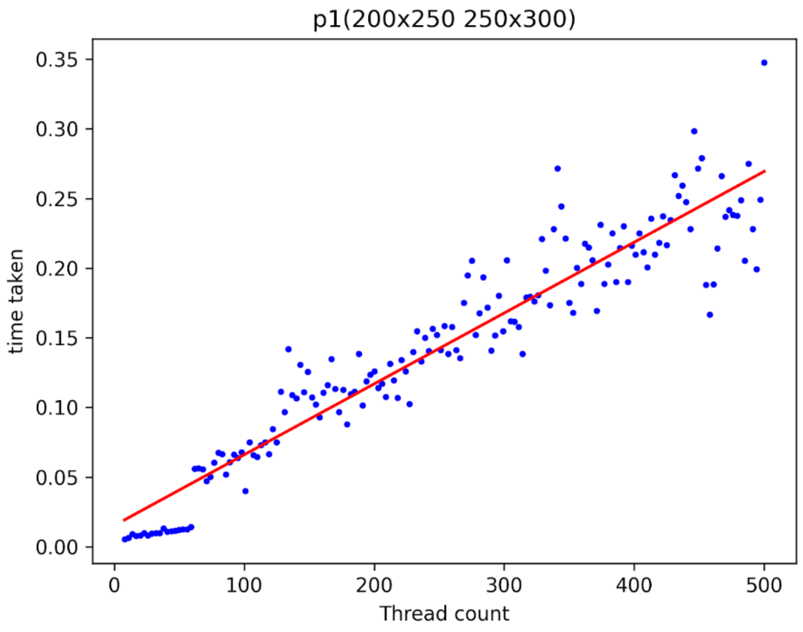
The assignment comprises of 4 parts. The file reading program (P1), matrix multiplying program (P2) and a scheduling program (S) and the analysis for context switching for time quanta of 1 ms and 2 ms.

Process-1:

In this process we use multi-threading to read a matrix from a text file. The process takes 5 inputs, dimensions of matrices (x, y, z), name of text file for matrix1 and name of text file for matrix2. Then the process spawn's 'n' threads to read each row of the matrix. We can vary the value of 'n' and optimize it such that the time required for reading the matrix is minimum.



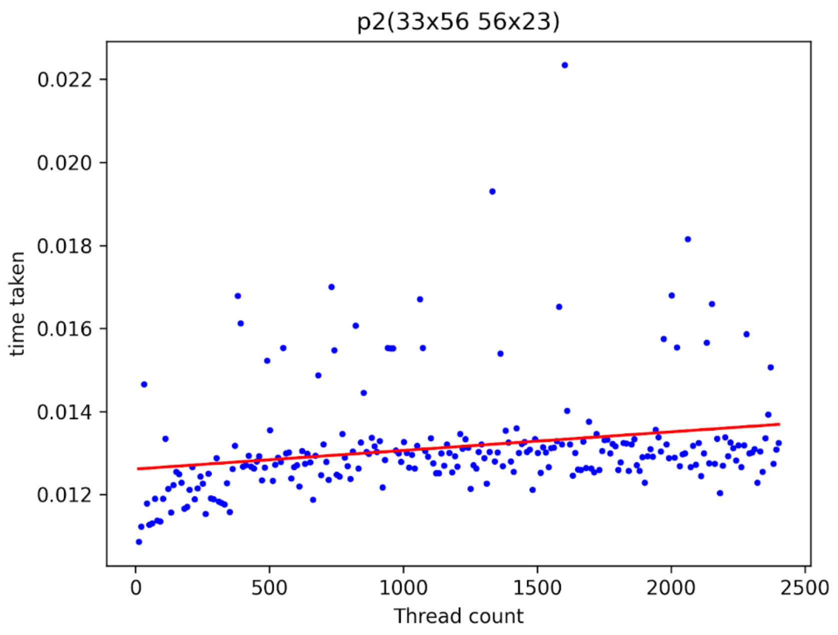


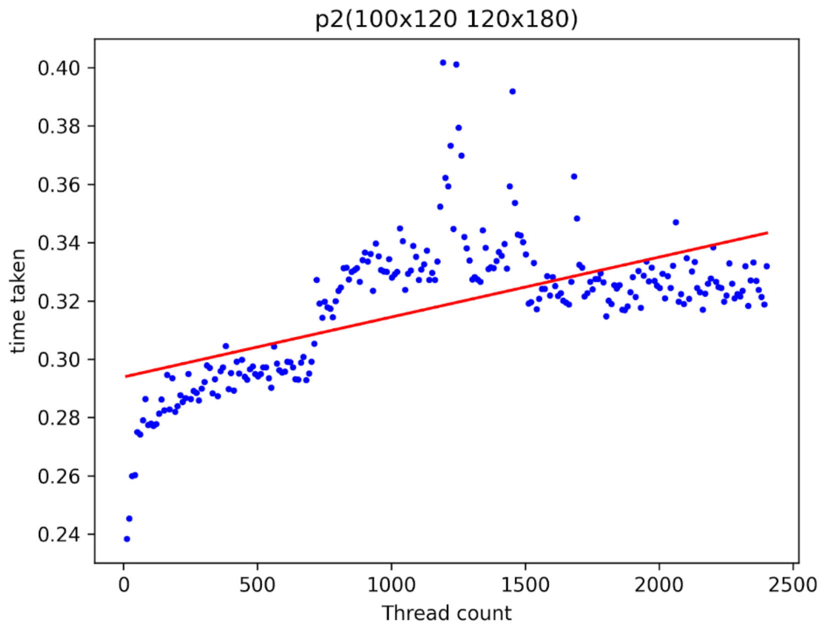


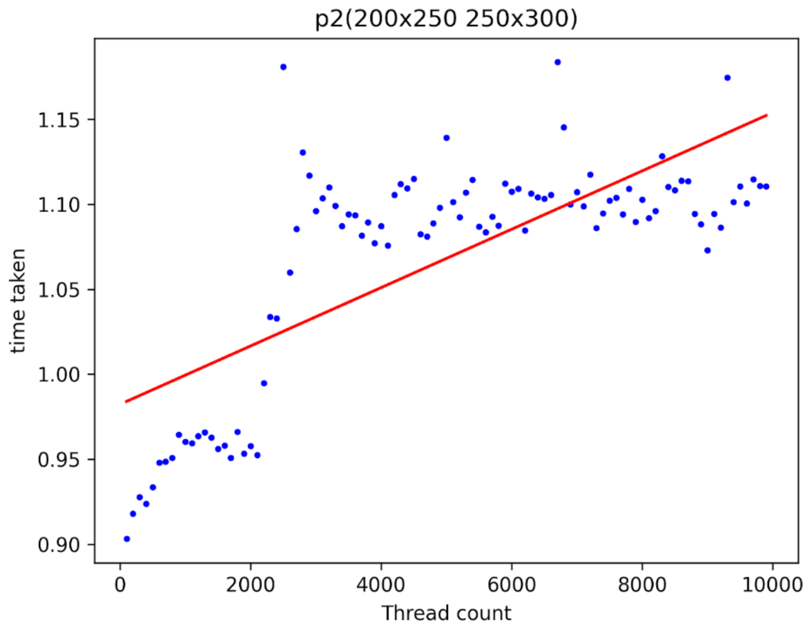
We have plotted the time against number of threads for 3 different sizes of matrices. As we can see the graphs have a positive slope. This is because as we increase the number of threads, the thread overhead increases but there is no effect on the time to read matrix as the CPU can run only a certain number (very few) of threads and the rest wait in the queue. As P1 spawn's 'n' threads, if 'n' is greater than the sum of number of rows in both matrix then some threads will not be used. We were expecting reading time to decrease till a certain number of threads (x+z) but because of I/O bound and thread overhead and the reasons mentioned above, we found the time to be increasing.

Process-2:

In this process we use multi-threading to multiply the matrices which are read by P1. We have used shared memory IPC mechanism to get the matrices read by P1 to P2. For matrix multiplication, a thread calculates each element of the final matrix by multiplying and adding the rows and column of matrix1 and matrix2 respectively. The process takes the name of the final matrix file as input. We can vary the number of threads to minimize the time required for the execution of program.







We have plotted the time against number of threads for 3 different sizes of matrices. As we can see the graphs have a positive slope. This is because as we increase the number of threads the thread overhead increases and the graph gets a positive slope. The maximum number of threads which can be used to multiply both the matrix is the number of elements present in the final matrix. We were expecting time to decrease till a certain number of threads (x*z) but after researching about the issue, we found that there can be many other factors like CPU usage, efficiency of memory allocation, etc. which can have negative effects on the program. As stated above for P1, CPU can handle very few threads at a time and hence the rest of the threads have to wait in the queue until a couple of CPU cycles. Therefore, due to thread overhead and restriction over the processors, a clear increase in the time taken can be seen with the increase in thread count.

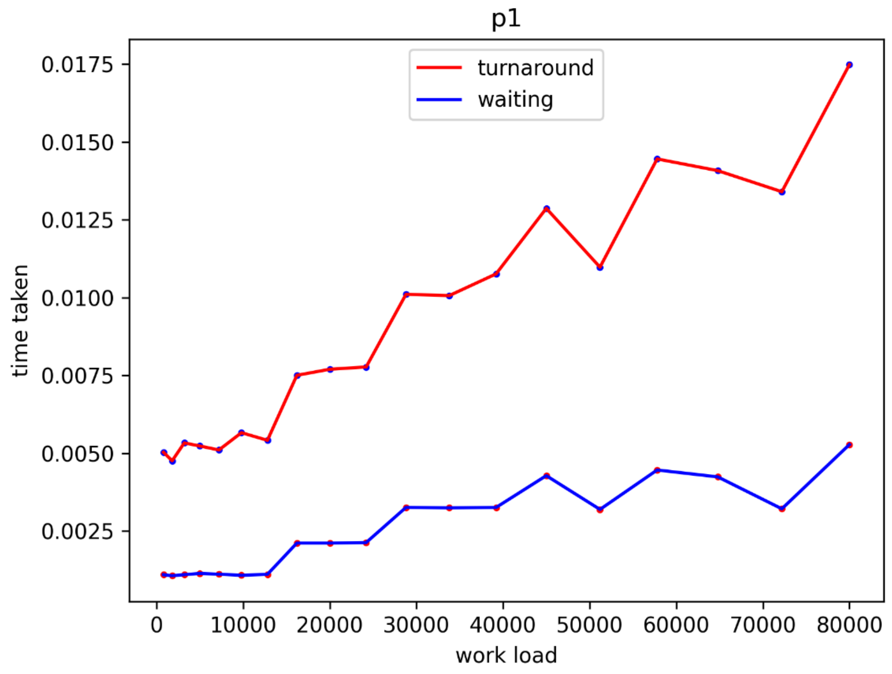
Scheduler(S):

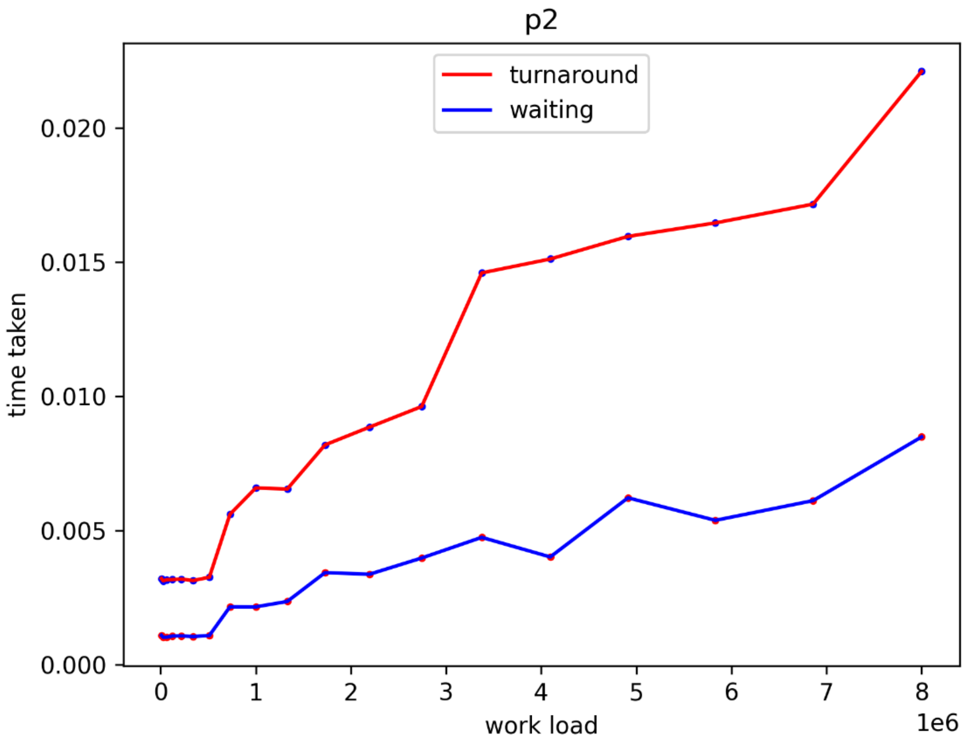
This is the scheduler process. It spawns P1 and P2 processes and controls their execution. We have created 6 signals (3 for each P1 and P2) with help of shared memory IPC which allows the scheduler to monitor and control P1 and P2. The signals that S uses are for pause/resume the processes, get the number of threads used by the processes and to get information if the process have been completed or not. With the help of the scheduler, we can schedule P1 and P2 in a round robin scheduling algorithm. We have plotted the graphs for turnaround time vs workload size and waiting time vs workload size for scheduler.

Round-Robin Algorithm:

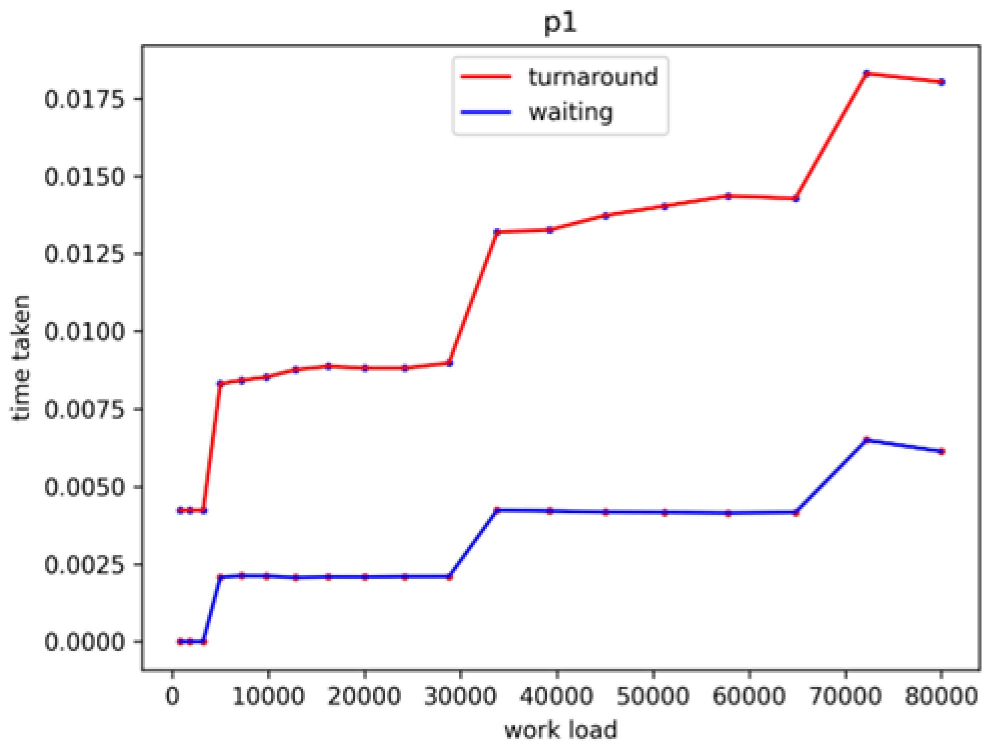
The scheduler assigns a fixed time quantum per process, and cycles through them. If process completes within that time-slice it gets terminated otherwise it is rescheduled after giving a chance to all other processes. RR scheduling involves extensive overhead, especially with a small time quantum. Good average response time, waiting time is dependent on number of processes, and not average process length. Starvation can never occur, since no priority is given. Order of time unit allocation is based upon process arrival time, similar to FIFO.

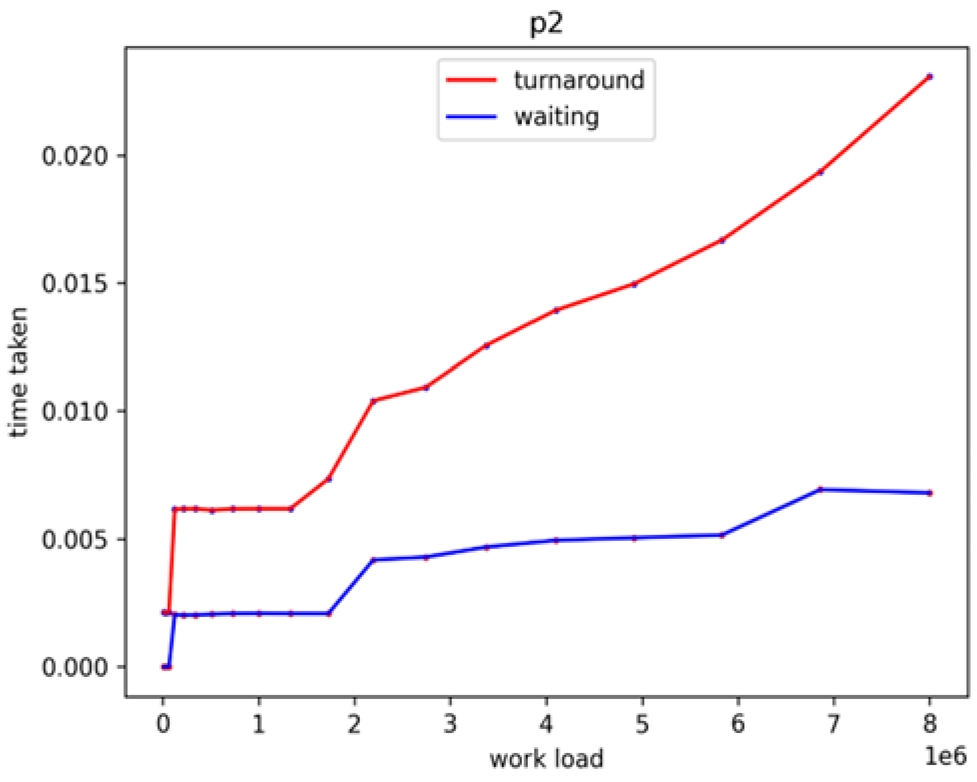
Round Robin for time Quantum 1ms:





Round Robin for time Quantum 2ms:





As we can see in the graph the total turnaround time and waiting time for any scheduling mechanism increases step wise because scheduler allots integral number of cycles for both the process P1 and P2. If a particular process finishes its task in between a certain slot, the scheduler still assumes that the process has not finished until the end of slot. So, the waiting time and turnaround time depends on number of cycles required to execute P1 and P2 and not the exact time. Also, another factor that plays a role in this is memory access which includes reading from and writing into a file and the accessing and altering of shared memory. With the increase in workload size, the number of memory accesses increases and also causes the total turnaround time to increase. The waiting time does not show a steep increase because if one process finishes it's execution (here, it is always P1 because P2 needs the data stored by P1 into the shared memory for computations), the other process (here, P2) will execute till the end without waiting. So the overall waiting time does not increase drastically.

Switching Overhead:

We have taken two 40x40 matrices for the program. With round robin scheduling for time quanta of 1 ms the switching over head is :0.000101398. Like-wise, round robin scheduling for time quanta of 2 ms the switching over head is: 0.000044059. As we can see the switching overhead is higher in case of time quanta 1 ms. This is because in case of 2 ms time quanta the number of contexts switching for P1 and P2 is less than that in case of 1 ms time quanta. So, the time taken for switching overhead is lower in 2 ms.