# OS Programming Assignment 2 – Report

**Thread Affinity**

## Design Of Program

I have designed a total of 4 programs each for chunks and mixed – to carry out both Experiment 1 and Experiment 2.

The general outline of my design is as follows

1. Taking input from the User file. I take N, K, C, and BT from the input file – which can be altered according to the needs of the user.
2. After I have taken the input, I create an array of my structure thread based on the value of 'k' entered. Then – I start to partition the rows for each – Chunk and Mixed – to proceed with the multiplication.

   After this is done – I proceed differently for each experiment as follows:

**Specifications of my machine:** My laptop has a total of 4 cores, and hence please use the value of C=4 while executing the program.

**Experiment 1:**
There are two kinds of programs in experiment 1 – one without threads being bound to cores(as in assignment 1), and the other with threads being bound to cores.

1. Without threads being bound to cores – Here I simply create the threads, and join them – without setting any affinity. The runner function carries out the multiplication of the rows for each thread, and in the end, I print the output into the out.txt file.
2. Threads being bound to cores – The approach here is slightly different. After creating the threads, I assign the first b threads to core1, the next b threads to core2, and so on until p*b number of threads is bounded. I do not do anything to the rest of the threads, and let the OS decide where to schedule them. In my program, j is the counter for the cores, and m is the counter for the b threads. Once j reaches b, we jump out of the loop.
   After this, the threads are joined, the output is printed into the outfile, and time is calculated.

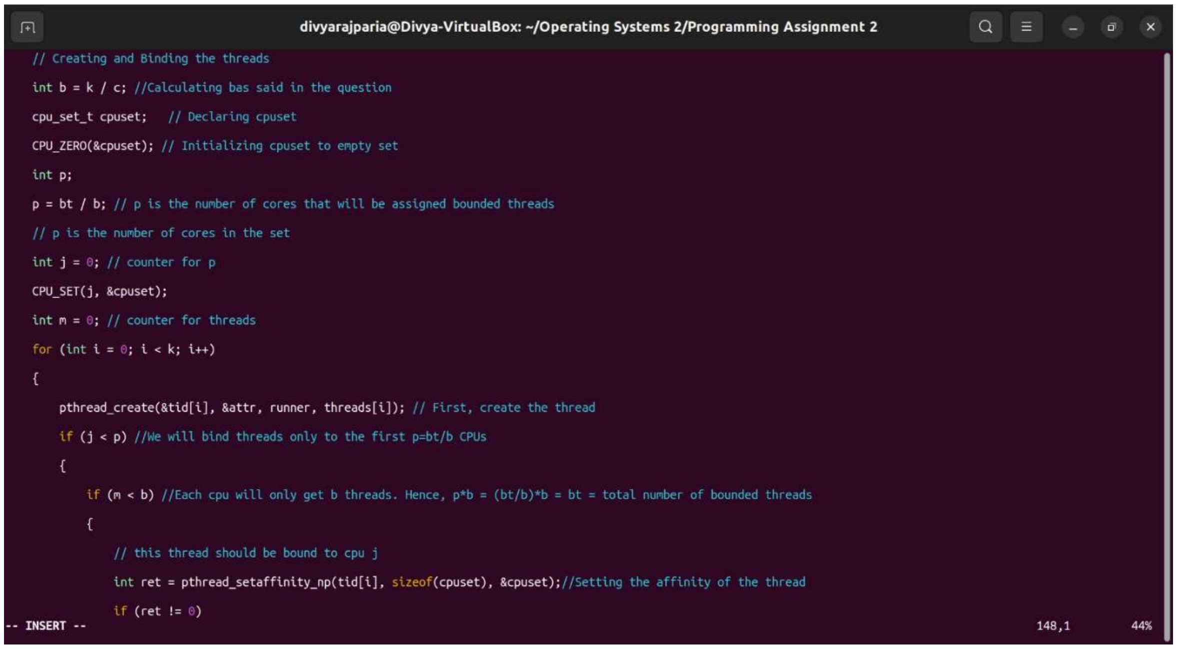
**Experiment 2:**

Here as well I have designed two programs, one with threads without being bound to cores, and one with threads being bound to cores

1. Average time of execution without threads being bound to cores – This program is exactly like program1 of experiment 1, except for the time calculation part. Here – we have to calculate the average time taken by each thread to finish its task. Hence – I have defined a global array of threadTimes – which stores the time when each thread executes its task in the runner function. At the end – I have calculated the sum of the times of each of the elements of the array and divided it by the total threads – which gives us the average time of execution of each thread.
2. Average time of execution with threads being bound to cores and Average time of execution of Normal Threads – I have done both of these in program 2 of experiment 2. The outline of this program is the same as others – except for the time calculation part. I have defined a global array of threadTimes – which stores time taken by each thread just like program 1. But – the first K/2 threads are bound to the first C/2 cores. The value of b remains the same here – as b = (k/2)/(c/2).
   avg_t1 calculates the average time taken by the bounded threads, and avg_t2 calculates the average time taken by the normal threads. The binding is done exactly like before.

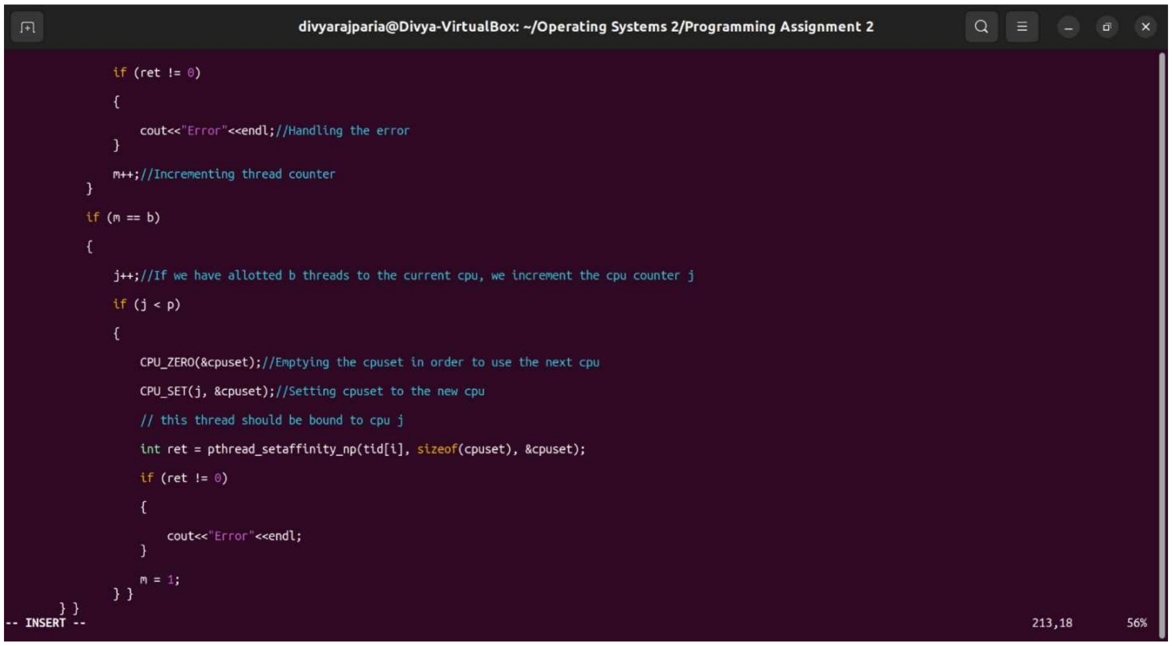
**Detailed Explanation of how I have Bound the Threads:**

I have attached pictures of my code – Along with comments on each line to explain that particular line. Since the main aim of this assignment was to see how bounded threads work, I have demonstrated my method of Binding the threads to the cores below:



divyarajparia@Divya-VirtualBox: ~/Operating Systems 2/Programming Assignment 2

```
// Creating and Binding the threads
int b = k / c; //Calculating bas said in the question
cpu_set_t cpuset;   // Declaring cpuset
CPU_ZERO(&cpuset); // Initializing cpuset to empty set
int p;
p = bt / b; // p is the number of cores that will be assigned bounded threads
// p is the number of cores in the set
int j = 0; // counter for p
CPU_SET(j, &cpuset);
int m = 0; // counter for threads
for (int i = 0; i < k; i++)
{
    pthread_create(&tid[i], &attr, runner, threads[i]); // First, create the thread
    if (j < p) //We will bind threads only to the first p=bt/b CPUs
    {
        if (m < b) //Each cpu will only get b threads. Hence, p*b = (bt/b)*b = bt = total number of bounded threads
        {
            // this thread should be bound to cpu j
            int ret = pthread_setaffinity_np(tid[i], sizeof(cpuset), &cpuset);//Setting the affinity of the thread
            if (ret != 0)
-- INSERT --                                                                                148,1         44%
```

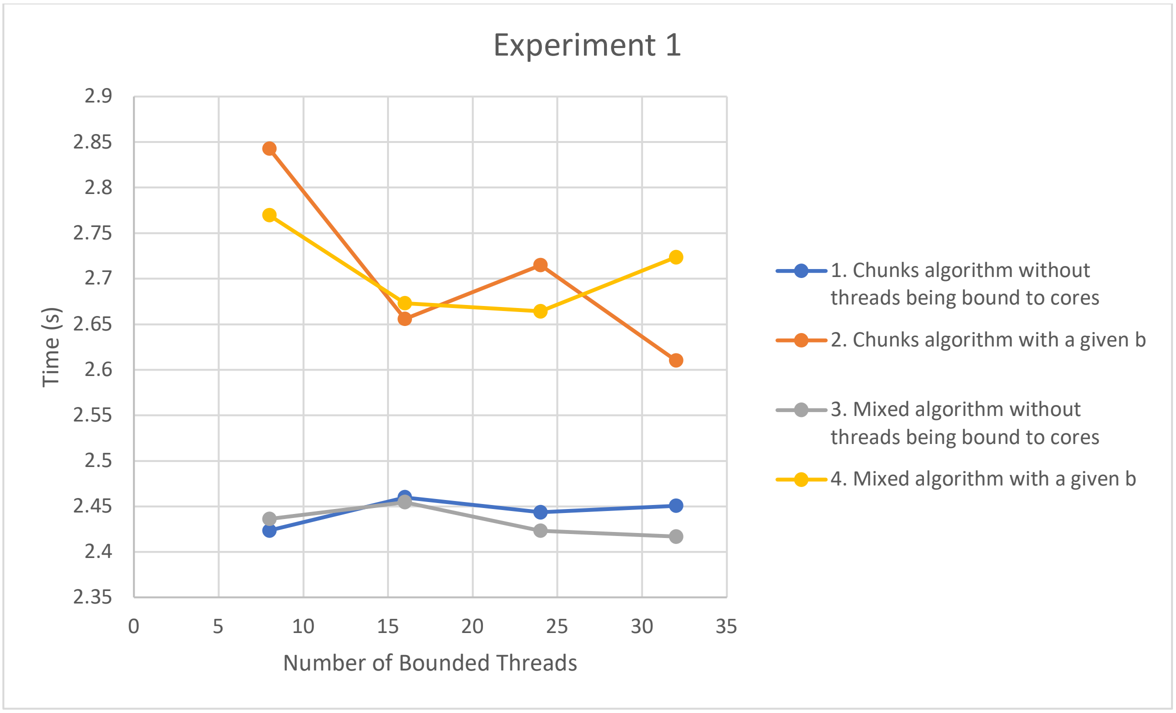


divyarajparia@Divya-VirtualBox: ~/Operating Systems 2/Programming Assignment 2

```
            if (ret != 0)
            {
                cout<<"Error"<<endl;//Handling the error
            }
            m++;//Incrementing thread counter
        }
        if (m == b)
        {
            j++;//If we have allotted b threads to the current cpu, we increment the cpu counter j
            if (j < p)
            {
                CPU_ZERO(&cpuset);//Emptying the cpuset in order to use the next cpu
                CPU_SET(j, &cpuset);//Setting cpuset to the new cpu
                // this thread should be bound to cpu j
                int ret = pthread_setaffinity_np(tid[i], sizeof(cpuset), &cpuset);
                if (ret != 0)
                {
                    cout<<"Error"<<endl;
                }
                m = 1;
            } }
    } }
-- INSERT --                                                                                213,18        56%
```

## Graphs

### Experiment 1:

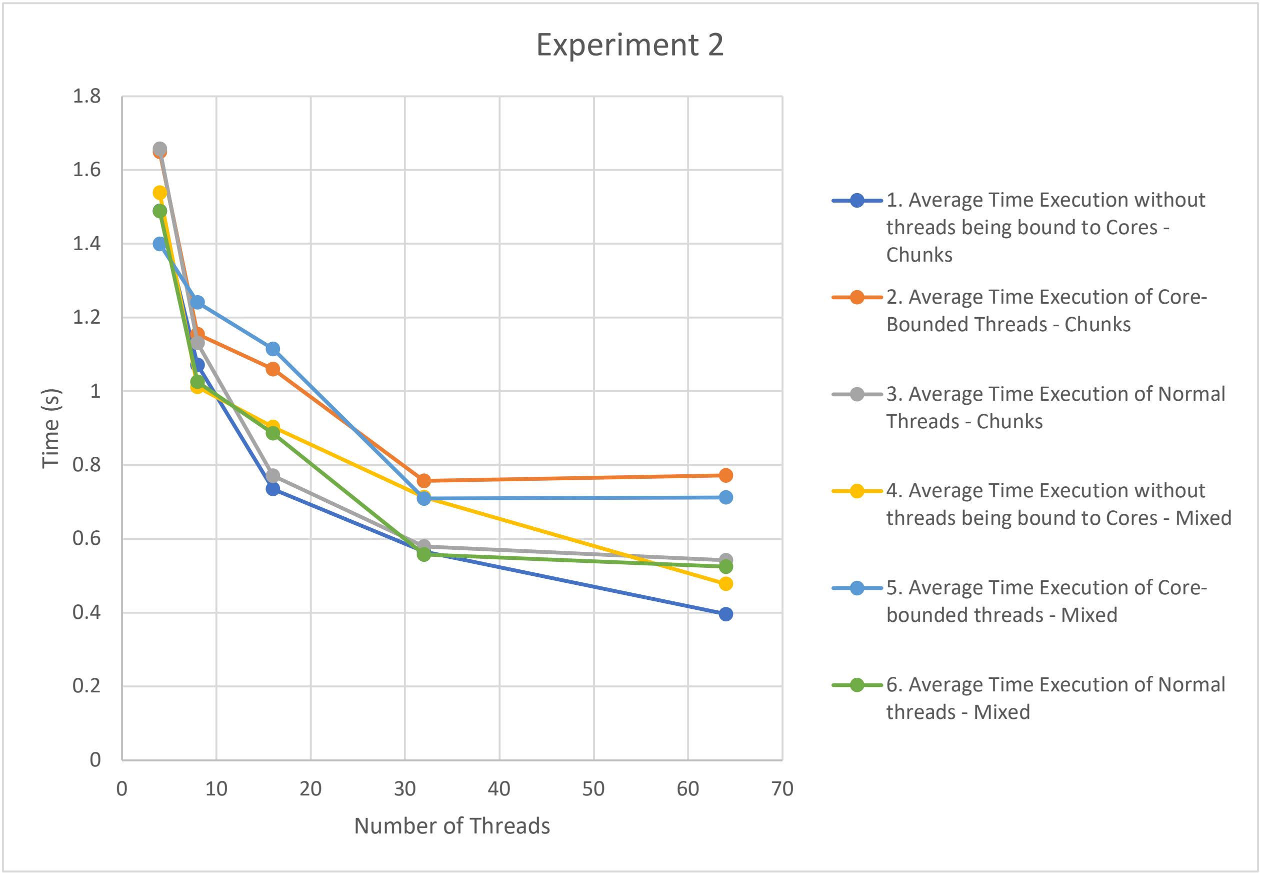


### Conclusion from the Graph:

1. Looking at the above graph, it is evident that in the case where we have not bound threads to the cores – performance is better. There is no improvement in performance after binding the threads to the cores.
2. On increasing the bounded threads from 8, 16, 24, 32 – there is no clear relationship between number of bounded threads and the performance time visible. On increasing the bounded threads, performance first improves, then deteriorates. It is probably because these experiments were conducted on a small matrix size of 1024, and hence the time taken to bind the threads was comparable to the time take to find the square of the matrix – and we do not see any improvement.

**Thus, there is no visible improvement in the performance time of the program after binding the threads to the cores.**

**Experiment 2:**



**Conclusions from the graph:**

1. Looking at the graphs, it is clear that on increasing the number of threads, the performance improves for every case.
2. However, we can notice that for both Chunks and Mixed algorithms, the average time of
   **Bound Threads > Normal Threads > Threads not being bound to cores**.
   Thus – we do not see any visible improvement in the performance of the algorithm by binding these threads to the cores.

**Thus, although the performance improves in each case on increasing the number of threads, we cannot see any performance advantage of binding the threads to the cores.**