

---

## Multi-Threading Matrix Multiplication

---

Name	Abdelrahman Ibrahim Gaber
ID	19015881
Department	Electronics and communication

# 1 Code Organization

The code is mainly using Multi-Threading technique using pthread.h library this library works in user level so it's not actually using a core for each thread but the library handle that and the threads works in parallel in such away that the library handle.

Here, Matrix Multiplication is a good example to show the concept of Multi-Threading since this program calculate the multiplication of 2 matrices that read it from 2 files in 3 ways each way is done in parallel with the others.

Method 1 : one Thread for whole matrix.

Method 2 : Thread per row.

Method 3 : Thread per each element.

Each matrix is setted as global pointer to pointer and be allocated in heap using dynamic memory allocation concepts by malloc function,

Main function gets files names that the user want to multiply the matrices including this files and start to read the input from the files and set it in its global pointer after allocated then main call the function that execute that 3 methods that we talked about every method has its routine function that passed to the thread function and start to execute the program in parallel according to number of threads used in every method. finally, every memory location we allocate it in the heap we should free it to prevent memory leakage.

## 2 Main functions

### 2.1 Read From File

```
void readFromFile(char *path, int fileNumber)
{
    FILE *f = fopen(path, READ_FROM_FILE);
    if(fileNumber == FIRST_FILE){
        fscanf(f , "row=%d col=%d",&rowsA , &colsA);
        allocateMatrix(f, fileNumber);
    }
    else{
        fscanf(f , "row=%d col=%d",&rowsB , &colsB);
        allocateMatrix(f, fileNumber);
    }
}
```

```

    }
    fclose(f);
}

```

### 2.1.2 Allocate Matrix

```

void allocateMatrix(FILE* fileName , int fileNumber){
    if(fileNumber == FIRST_FILE){
        matA = (long **)malloc(rowsA * sizeof(long **));
        for(int i = 0 ; i < rowsA ; i++){
            matA[i] = (long *)malloc(colsA * sizeof(long *));
            for(int j = 0 ; j < colsA ; j++){
                fscanf(fileName , "%ld" , &matA[i][j]);
            }
            fgetc(fileName);
        }
    }
    else{
        matB = (long **)malloc(rowsB * sizeof(long **));
        for(int i = 0 ; i < rowsB ; i++){
            matB[i] = (long *)malloc(colsB * sizeof(long *));
            for(int j = 0 ; j < colsB ; j++){
                fscanf(fileName , "%ld" , &matB[i][j]);
            }
            fgetc(fileName);
        }
    }
}

```

### 2.2 Heap Allocator

```

void heapAllocator(void){
    matC_perEle = (long **)malloc(sizeof(long *) * rowsA);
    matC_perRow = (long **)malloc(sizeof(long*) * rowsA);
    matC_whole = (long **)malloc(sizeof(long *) * rowsA);
    for(int i = 0 ; i < rowsA ; i++){

```

```

        matC_perEle[i] = (long *)malloc(sizeof(long) * colsB);
        matC_perRow[i] = (long *)malloc(sizeof(long) * colsB);
        matC_whole[i] = (long *)malloc(sizeof(long) * colsB);
    }

}

}

2.3.1 Run case 1

void runCase1(void){
    struct timeval stop , start;
    gettimeofday(&start , NULL);
    pthread_t threadPerWholeMatrix ;
    /*one thread for the whole matrix no arguments*/
    if(pthread_create(&threadPerWholeMatrix , NULL , &mutrixMul , NULL)!= 0)
        perror("Error creating thread\n");
        exit(EXIT_FAILURE);
    }
    pthread_join(threadPerWholeMatrix , NULL);
    gettimeofday(&stop , NULL);
    printf("Thread per matrix taken in Micro Second %lu\n",stop.tv_usec - start.tv_usec);
    printf("Threads Created = 1\n");
}
}

```

### 2.3.2 Run case 2

```

void runCase2(void){
    struct timeval stop , start ;
    gettimeofday(&start , NULL);
    pthread_t threadPerRow[rowsA];
    for(int i = 0 ; i < rowsA ; i++){
        int* arg = malloc(sizeof(int)*20);
        *arg = i;
        if(pthread_create(&threadPerRow[i] , NULL , &mutrixMulPerRow , arg)

```

```

        perror("Error creating thread\n");
        exit(EXIT_FAILURE);
    }
}

/*wait for all threads being created*/
for(int i = 0 ; i < rowsA ; i++){
    pthread_join(threadPerRow[i] , NULL);
}

gettimeofday(&stop , NULL);
printf("Thread per Row taken in Micro Second %lu\n",stop.tv_usec - start.tv_usec);
printf("Threads Created = %d\n",rowsA);
}

}

```

### 2.3.3 Run case 3

```

void runCase3(void){
    struct timeval stop , start;
    gettimeofday(&start , NULL);
    pthread_t threadPerElement[rowsA * colsB];
    int threadIndex = 0 ;
    for(int i = 0 ; i < rowsA ; i++){
        for(int j = 0 ; j < colsB ; j++){
            matData *args = malloc(sizeof(matData));
            args->currentRow = i;
            args->currentColoumn = j;
            if(pthread_create(&threadPerElement[threadIndex++] , NULL ,
                &matrixMulPerElement , args)){
                perror("Error creating thread\n");
                exit(EXIT_FAILURE);
            }
        }
    }

    for(int i = 0 ; i < rowsA * colsB; i++){
        pthread_join(threadPerElement[i] , NULL);
    }
}

```

```

    }
    gettimeofday(&stop , NULL);
    printf("Thread per Element taken in Micro Second %lu\n",stop.tv_usec - s
    printf("Threads Created = %d\n",(rowsA * colsB));
}
}

```

## 2.4 Output Handler

```

void outputHandler(int argc , char*argv[]){
    if(argc == 1){
        writeInFile("c_per_matrix.txt",METHOD_1);
        writeInFile("c_per_row.txt",METHOD_2);
        writeInFile("c_per_element.txt",METHOD_3);
    }
    else{
        char*matrix_1 = malloc(sizeof(char) * 20);
        char*matrix_2 = malloc(sizeof(char) * 20);
        char*matrix_3 = malloc(sizeof(char) * 20);
        strcpy(matrix_1 , argv[3]);
        strcat(matrix_1 , "_per_matrix.txt");
        writeInFile(matrix_1 , METHOD_1);
        free(matrix_1);
        strcpy(matrix_2 , argv[3]);
        strcat(matrix_2 , "_per_row.txt");
        writeInFile(matrix_2 , METHOD_2);
        free(matrix_2);
        strcpy(matrix_3 , argv[3]);
        strcat(matrix_3 , "_per_element.txt");
        writeInFile(matrix_3 , METHOD_3);
        free(matrix_3);
    }
}
}
}

```

## 2.5 Lite Garbage Collector

```

void liteGarbageCollector(void){
    /*free pointers that take place in heap*/
    for(int i = 0 ; i < rowsA ; i++){
        free(matA[i]);
        free(matC_whole[i]);
        free(matC_perRow[i]);
        free(matC_perEle[i]);
    }
    for(int i = 0 ; i < rowsB ; i++) free(matB[i]);
    free(matA);
    free(matB);
    free(matC_whole);
    free(matC_perRow);
    free(matC_perEle);
}
}

```

### 3 How to compile this code

3.1 This program simply can be run from the terminal using this command `gcc fileName.c -o fileName -lpthread`

3.2 `-lpthread` is required to enable the operating system to run in Multi-Threading way.

3.3 then there is 3 cases you can run

3.3.1 case 1 : `./fileName` : run using the default files `a.txt` , `b.txt` and write the ouput in `c.txt` for each method.

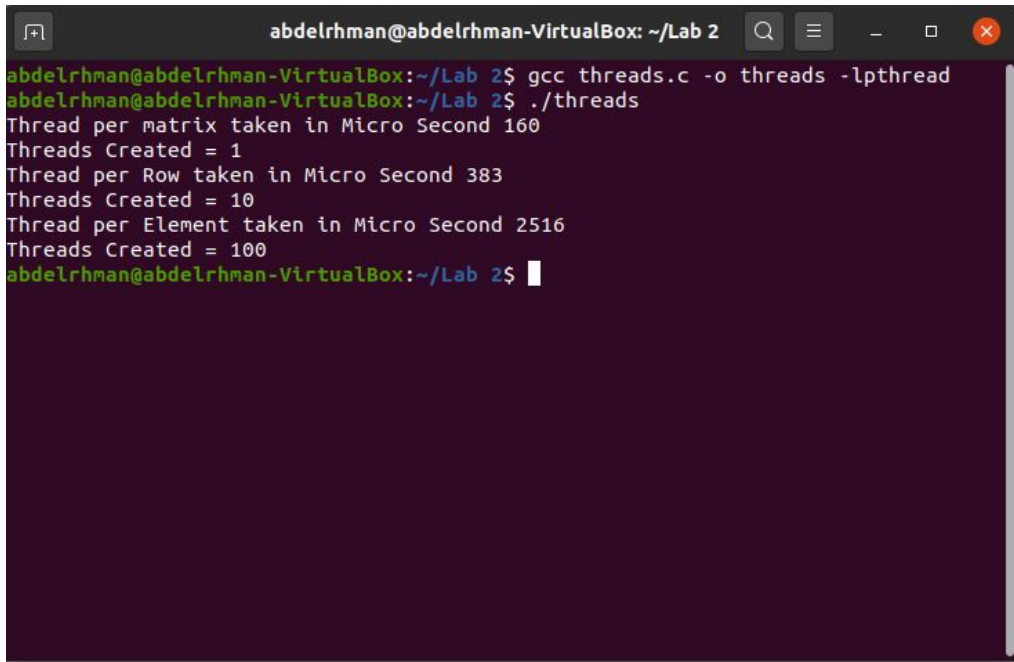
3.3.2 case 2 : `./fileName a b c` : `a` and `b` are such arguments that are also considered the default and the program handle that case and write the ouput in `c.txt`.

3.3.3 case 3 : `./fileName x y z` : custom inputs and the program consider `x` and `y` the 2 inputs files that multiply the 2 matrices on it and write the result in the last argument `z.txt`

3.4 Now the user can read the output from the certain file that he choose

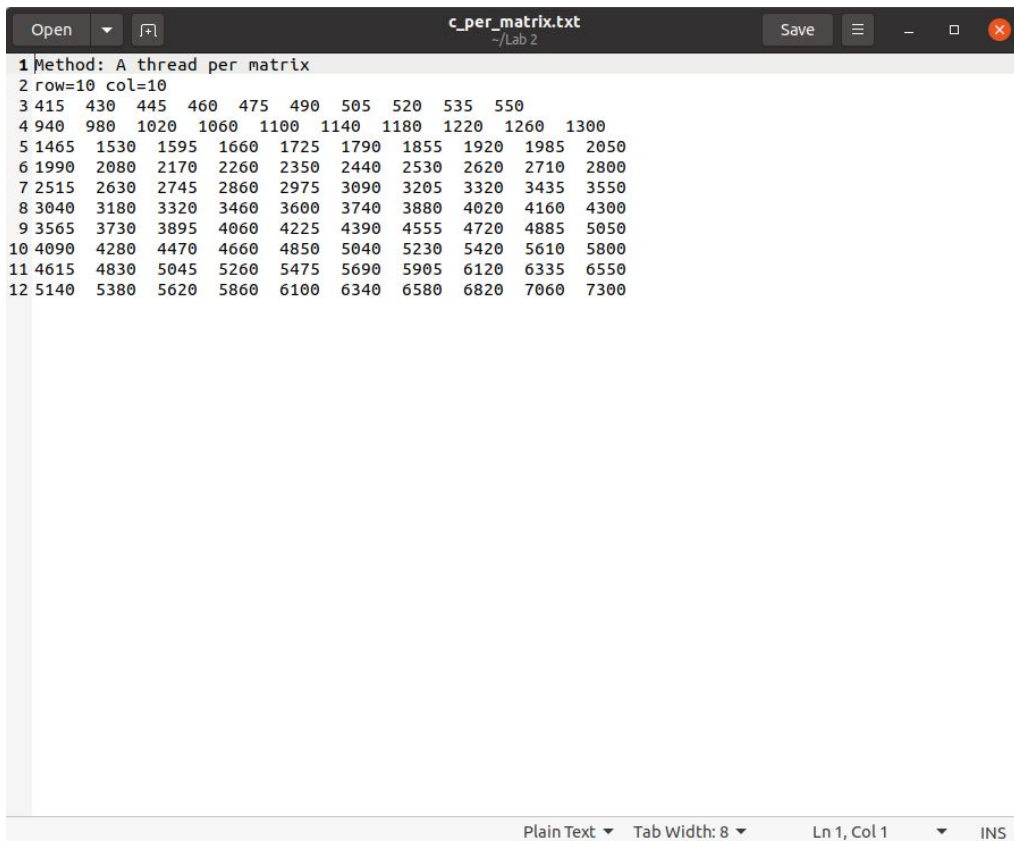
## 4 Sample Run

### 4.1 Test case 1 (No Arguments)



```
abdelrhman@abdelrhman-VirtualBox: ~/Lab 2
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ gcc threads.c -o threads -lpthread
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ ./threads
Thread per matrix taken in Micro Second 160
Threads Created = 1
Thread per Row taken in Micro Second 383
Threads Created = 10
Thread per Element taken in Micro Second 2516
Threads Created = 100
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$
```

Figure 1: compile test 1 with no argument



```
c_per_matrix.txt
~/Lab 2
1 Method: A thread per matrix
2 row=10 col=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 1260 1300
5 1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
6 1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
7 2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
8 3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
9 3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
10 4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
11 4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
12 5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
```

Figure 2: one thread per matrix



```
Open [icon] c_per_row.txt [Save] [Menu] [Window] [Close]
~/Lab 2
1 Method: A thread per row
2 row=10 col=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 1260 1300
5 1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
6 1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
7 2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
8 3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
9 3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
10 4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
11 4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
12 5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
```

Figure 3: one thread per row

```
Open [icon] c_per_element.txt [Save] [Menu] [Window] [Close]
~/Lab 2
1 Method: A thread per element
2 row=10 col=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 1260 1300
5 1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
6 1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
7 2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
8 3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
9 3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
10 4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
11 4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
12 5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
```

Plain Text Tab Width: 8 Ln 1, Col 1 INS

Figure 4: one thread per element

## 4.2 test case 2 (arguments)

```
abdelrhman@abdelrhman-VirtualBox: ~/Lab 2
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ ./threads a b c
Thread per matrix taken in Micro Second 194
Threads Created = 1
Thread per Row taken in Micro Second 581
Threads Created = 10
Thread per Element taken in Micro Second 2609
Threads Created = 100
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$
```

Figure 5: compile test 2 with no argument

```
c_per_matrix.txt
~/Lab 2
1 Method: A thread per matrix
2 row=10 col=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 1260 1300
5 1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
6 1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
7 2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
8 3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
9 3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
10 4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
11 4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
12 5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
```

Figure 6: one thread per matrix

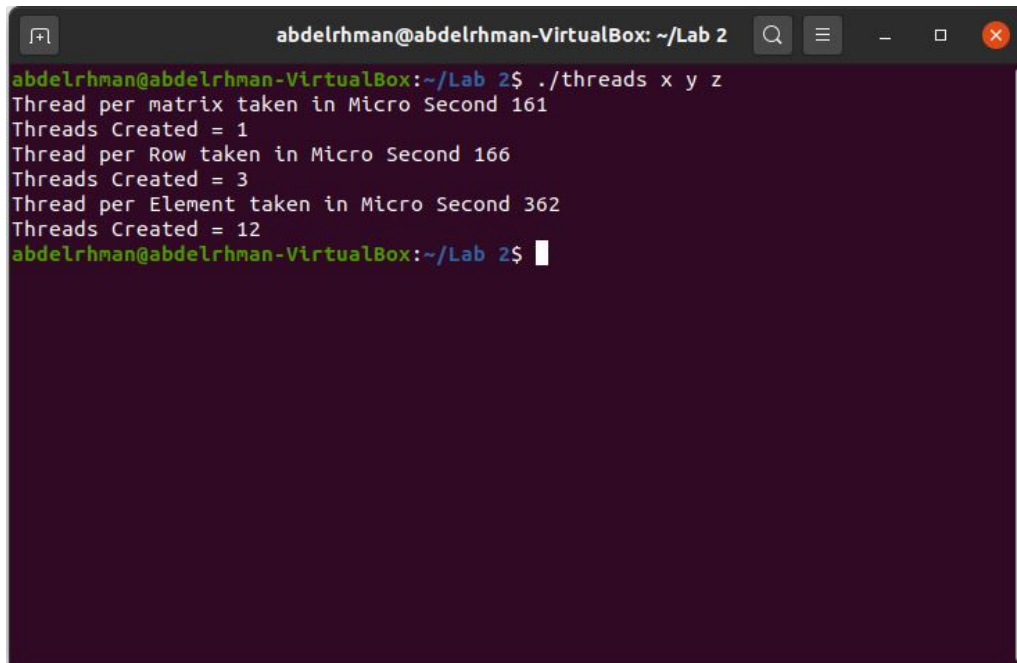
```
c_per_row.txt
~/Lab 2
Open Save
1 Method: A thread per row
2 row=10 col=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 1260 1300
5 1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
6 1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
7 2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
8 3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
9 3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
10 4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
11 4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
12 5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
```

Figure 7: one thread per row

```
c_per_element.txt
~/Lab 2
Open Save
1 Method: A thread per element
2 row=10 col=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 1260 1300
5 1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
6 1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
7 2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
8 3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
9 3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
10 4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
11 4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
12 5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
Plain Text Tab Width: 8 Ln 1, Col 1 INS
```

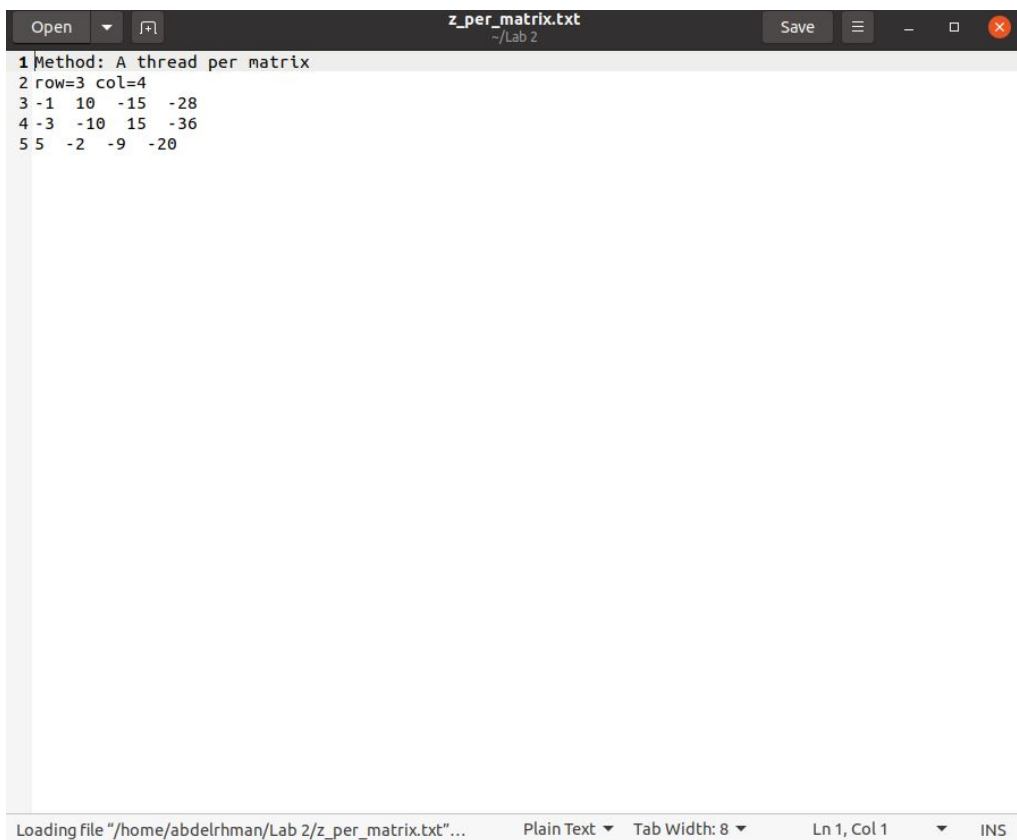
Figure 8: one thread per element

#### 4.3.1 test case 3 (Custom inputs)



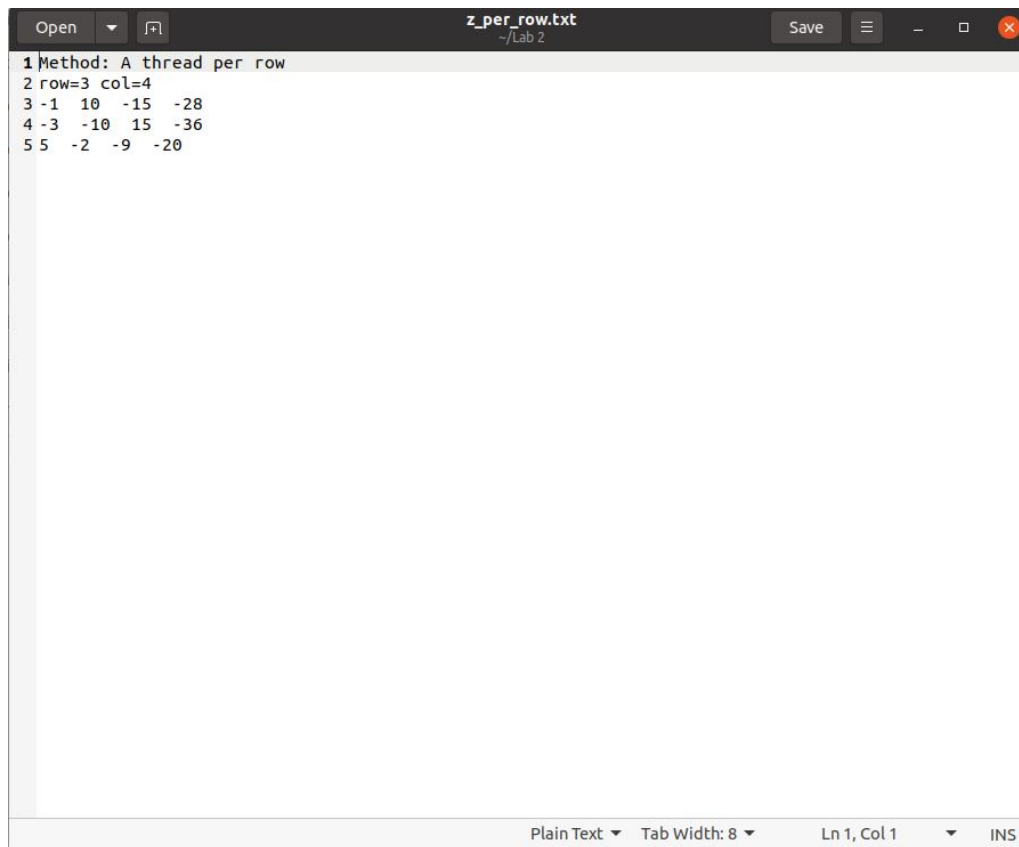
```
abdelrhman@abdelrhman-VirtualBox: ~/Lab 2
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ ./threads x y z
Thread per matrix taken in Micro Second 161
Threads Created = 1
Thread per Row taken in Micro Second 166
Threads Created = 3
Thread per Element taken in Micro Second 362
Threads Created = 12
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$
```

Figure 9: compile test 3 with no argument



```
z_per_matrix.txt
1 Method: A thread per matrix
2 row=3 col=4
3 -1 10 -15 -28
4 -3 -10 15 -36
5 5 -2 -9 -20
```


Figure 10: one thread per matrix



```
1 Method: A thread per row
2 row=3 col=4
3 -1 10 -15 -28
4 -3 -10 15 -36
5 5 -2 -9 -20
```

Plain Text Tab Width: 8 Ln 1, Col 1 INS

Figure 11: one thread per row



```
1 Method: A thread per element
2 row=3 col=4
3 -1 10 -15 -28
4 -3 -10 15 -36
5 5 -2 -9 -20
```

Loading file "/home/abdelrhman/Lab 2/z\_per\_element.txt"... Plain Text Tab Width: 8 Ln 1, Col 1 INS

Figure 12: one thread per element

#### 4.3.2 another custom inputs

```
abdelrhman@abdelrhman-VirtualBox: ~/Lab 2
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ ./threads g h k
Thread per matrix taken in Micro Second 122
Threads Created = 1
Thread per Row taken in Micro Second 190
Threads Created = 5
Thread per Element taken in Micro Second 516
Threads Created = 20
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$
```

Figure 13: compile test 3 with no argument

```
k_per_matrix.txt
~/Lab 2
1 Method: A thread per matrix
2 row=5 col=4
3 175 190 205 220
4 400 440 480 520
5 625 690 755 820
6 850 940 1030 1120
7 1075 1190 1305 1420
```

Loading file "/home/abdelrhman/Lab 2/k\_per\_matrix.txt"... Plain Text Tab Width: 8 Ln 1, Col 1 INS

Figure 14: one thread per matrix

```
1 Method: A thread per row
2 row=5 col=4
3 175 190 205 220
4 400 440 480 520
5 625 690 755 820
6 850 940 1030 1120
7 1075 1190 1305 1420
```

Figure 15: one thread per row

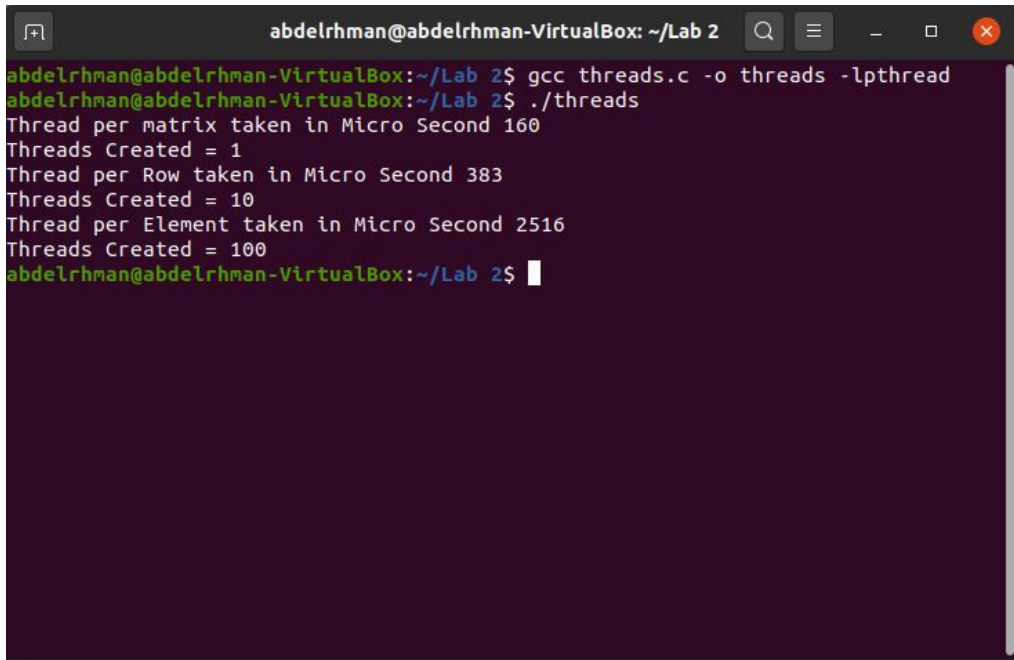
```
1 Method: A thread per element
2 row=5 col=4
3 175 190 205 220
4 400 440 480 520
5 625 690 755 820
6 850 940 1030 1120
7 1075 1190 1305 1420
```

Figure 16: one thread per element

It's noticeable that the result in each method to multiply is same because each of them do the same functionality but in different number of threads

## 5 Comparison between three methods

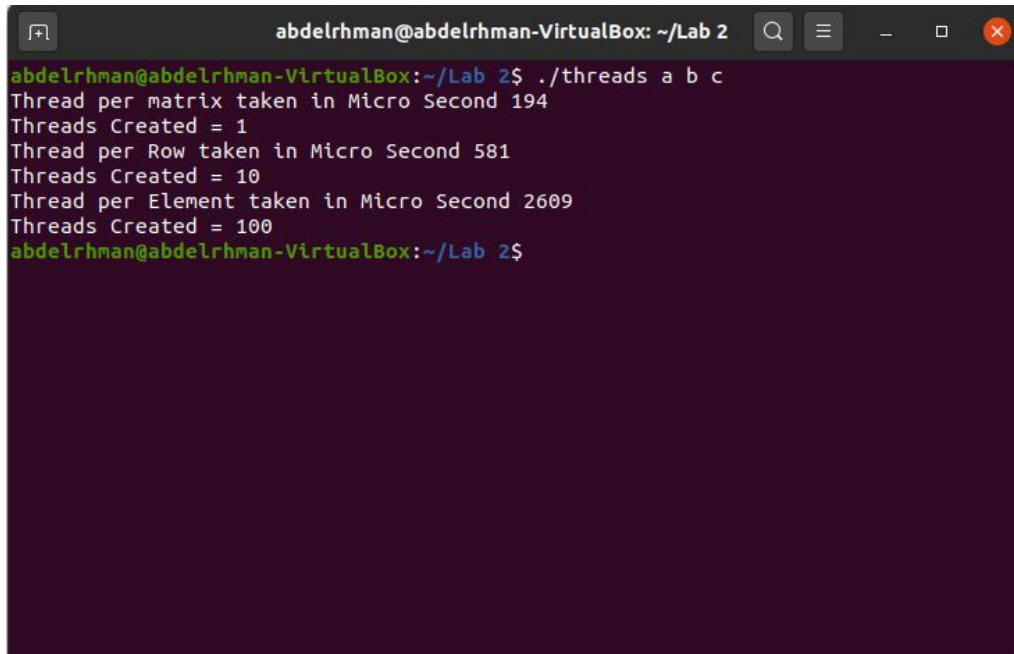
### 5.1 time of execution in test 1



```
abdelrhman@abdelrhman-VirtualBox: ~/Lab 2
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ gcc threads.c -o threads -lpthread
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ ./threads
Thread per matrix taken in Micro Second 160
Threads Created = 1
Thread per Row taken in Micro Second 383
Threads Created = 10
Thread per Element taken in Micro Second 2516
Threads Created = 100
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$
```

Figure 17: Time of execution

### 5.2 time of execution in test 2

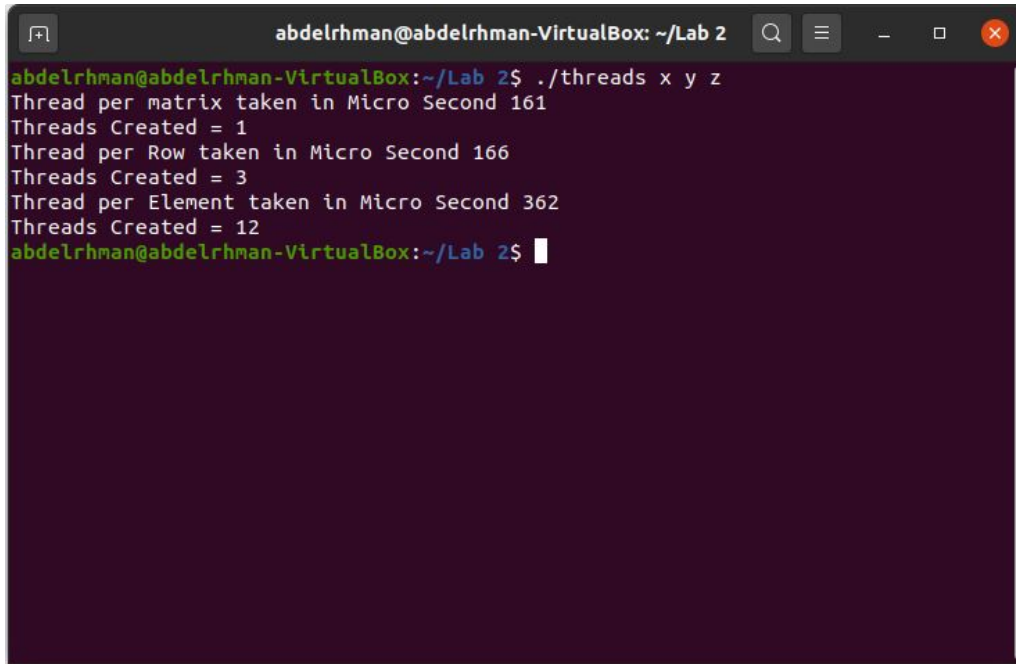


```
abdelrhman@abdelrhman-VirtualBox: ~/Lab 2
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ ./threads a b c
Thread per matrix taken in Micro Second 194
Threads Created = 1
Thread per Row taken in Micro Second 581
Threads Created = 10
Thread per Element taken in Micro Second 2609
Threads Created = 100
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$
```

Figure 18: Time of execution

### 5.3 time of execution in test 3



A terminal window titled 'abdelrhman@abdelrhman-VirtualBox: ~/Lab 2' with search, menu, and window control icons. The terminal shows the execution of a script './threads x y z'. The output displays three methods: Method 1 (1 thread, 161 microseconds), Method 2 (3 threads, 166 microseconds), and Method 3 (12 threads, 362 microseconds). The prompt 'abdelrhman@abdelrhman-VirtualBox:~/Lab 2\$' is visible at the bottom.

```
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$ ./threads x y z
Thread per matrix taken in Micro Second 161
Threads Created = 1
Thread per Row taken in Micro Second 166
Threads Created = 3
Thread per Element taken in Micro Second 362
Threads Created = 12
abdelrhman@abdelrhman-VirtualBox:~/Lab 2$
```

Figure 19: Time of execution

In this figures we can notice that the methods are different in time execution.

- Method 1 : the fastest and uses only 1 thread
- Method 2 : faster from method 3 and slower than method 1 since uses thread per row
- Method 3 : the slowest method since uses thread per element

The number of threads mainly have the upper hand in calculate the execution time since the switching between threads takes time to switch between thread context (pc,register,etc) so according to that :-

- Method 1 : only have one thread so it is the fastest
- Method 2 : have threads bigger than method 1 and bigger than method 2 so its speed in between
- Method 3 : have the biggest number of threads , a thread per element so it's clearly that it's the slowest method