Name: Mahmoud Ahmed Ibrahim Ahmed ID: 1901530

Lab2: Multithreading

The main consists of 5 parts:

Part 1: Handling the arguments.

```c
int main(int argc, char *argv[])
{
    /* Handling The arguments */
    char a = 'a';
    char b = 'b';
    char c = 'c';
    if (argc > 1)
    {
        a = *(argv[1]);
        b = *(argv[2]);
        c = *(argv[3]);
    }
```

The three names of the 2 input files and output files are initialized by a, b, c. but if the user pass the 3 names as arguments while running the code these variables will be updated

Part 2: Read the input matrices from .txt files

```c
/* Read the input matrices from the txt files */
Matrix_t A, B, C;
readInputFiles(&A, a);
readInputFiles(&B, b);
```

Read Input Files function called twice to read the two input matrices and assign them to A and B

Part 3, 4 and 5: multiply the two matrices and put the result in the output files. First, current time assigned to start, then multiply matrix function is called giving it enum argument PER_MATRIX or PER_ROW or PER_ELEMENT, then current time assign to stop so difference between them is the execution time of multiplication. Finally, writing these outputs to the output file then free the allocated space.

```
//////////////////////Per Matrix//////////////////////

/* Start The timer */
printf("/////////////////Per Matrix Time/////////////////\n");
gettimeofday(&start, NULL);
C = multiplyMatrix(&A, &B, PER_MATRIX);
gettimeofday(&stop, NULL);
/* Stop the timer */

/* Print The time */
printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);
writeOutputFiles(&C, c, PER_MATRIX);
for (i = 0; i < C.row; i++)
    free(C.mat[i]);
free(C.mat);
```

```
//////////////////////Per ROW//////////////////////

/* Start The timer */
printf("/////////////////Per ROW Time/////////////////\n");
gettimeofday(&start, NULL);
C = multiplyMatrix(&A, &B, PER_ROW);
gettimeofday(&stop, NULL);
/* Stop the timer */

/* print the time */
printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);
writeOutputFiles(&C, c, PER_ROW);
for (i = 0; i < C.row; i++)
    free(C.mat[i]);
free(C.mat);

//////////////////////Per ELEMENT//////////////////////

/* Start The timer */
printf("///////////////Per ELEMENT Time///////////////\n");
gettimeofday(&start, NULL);
C = multiplyMatrix(&A, &B, PER_ELEMENT);
gettimeofday(&stop, NULL);
/* Stop the timer */

/* print the time */
printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);
writeOutputFiles(&C, c, PER_ELEMENT);
for (i = 0; i < C.row; i++)
    free(C.mat[i]);
free(C.mat);
```

The code consists of 7 main functions:

1- Read input files which takes pointer to matrix and file name, it opens the file and assign the values to that matrix
2- Multiply matrix which takes two pointers to two matrices and return the result of their multiplication it also takes it enum argument which could be PER_MATRIX or PER_ROW or PER_ELEMENT to decide which method to use in calculations.
3- Multiply row and column which is helping function takes pointer to row and pointer to column and the length of them and return the result of their multiplication.
4- Per row multiply which is the function passed to the thread creation as it calculates certain row of the result, it takes void pointer argument which is pointer to struct arguments typecasted to void pointer, this struct contains pointer to the resulted matrix to write in it, pointer to second matrix transposed, pointer to current row of the first matrix and its length.
5- Per element multiply which is the function passed to the thread creation as it calculates certain element of the result it takes void pointer argument which is pointer to struct ArgumentsPerElements typecasted to void pointer, this struct contains pointer to the resulted matrix to write in it, pointer to current row of the first matrix, pointer to the current column of the second matrix and their length.
6- Transpose which returns the transpose of the given matrix so that multiplication becomes easier.
7- Write output to files which write the contents of the given matrix (output matrix) it takes enum argument which could be PER_MATRIX or PER_ROW or PER_ELEMENT to decide which file to write in.

Compiling and Running

The code could be compiled by writing following line in terminal.

gcc main.c -o a -lm

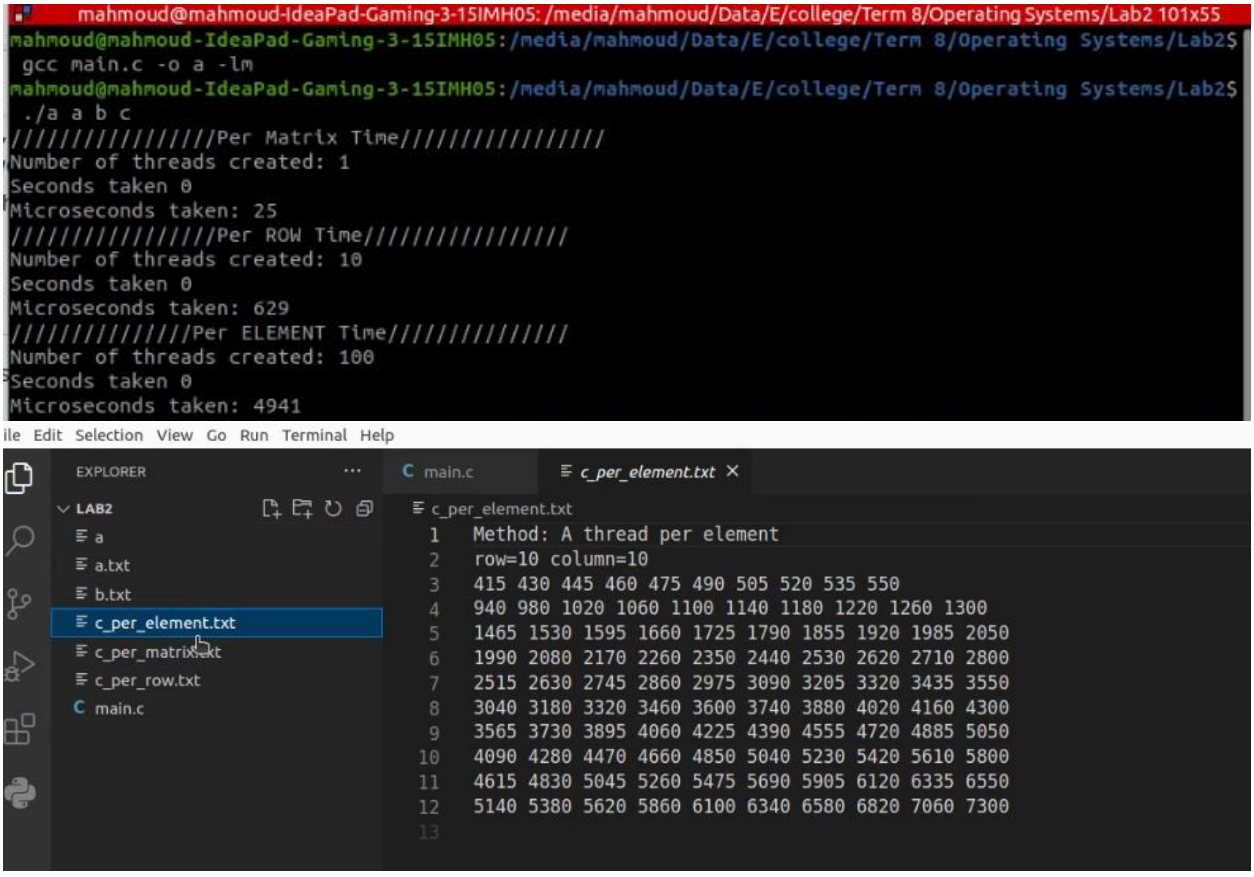
where main.c is the name of the source C file and a is the output executable file, then we can run by following line in terminal.

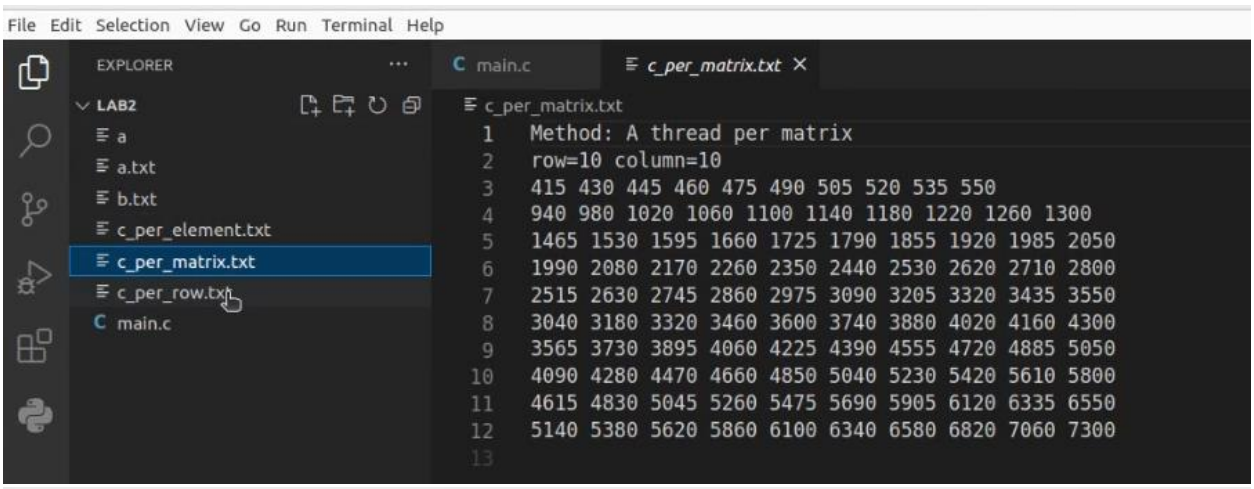
./a a b c

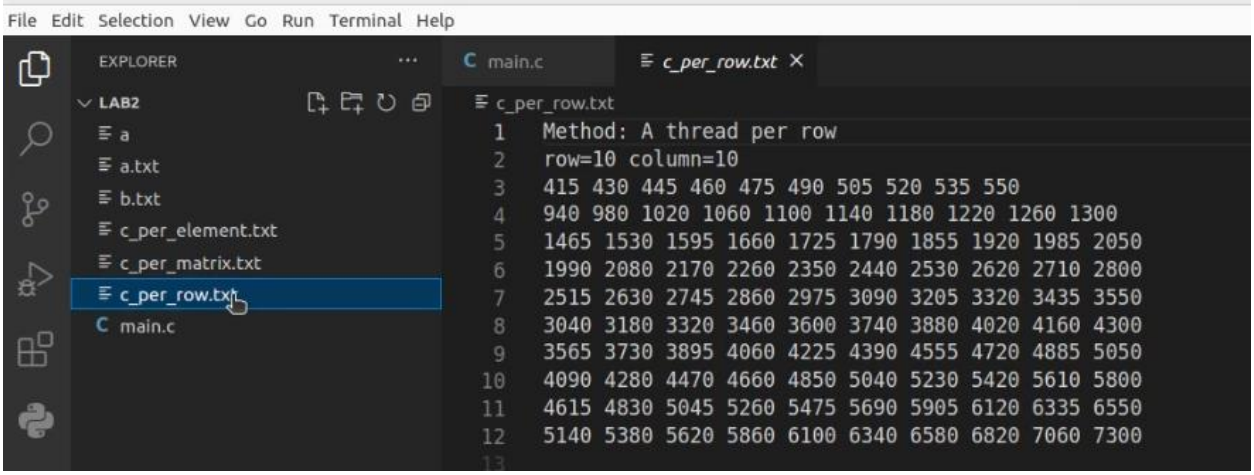
Where a, b are the names of the input files and c for output files, if nothing written the default names are a, b and c.

Sample Runs

Test 1

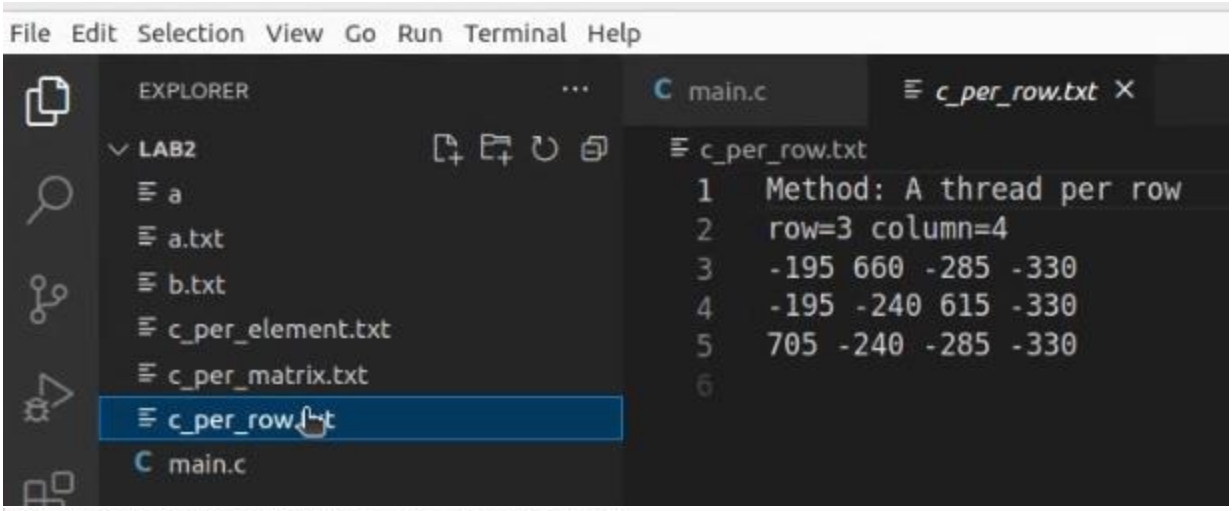


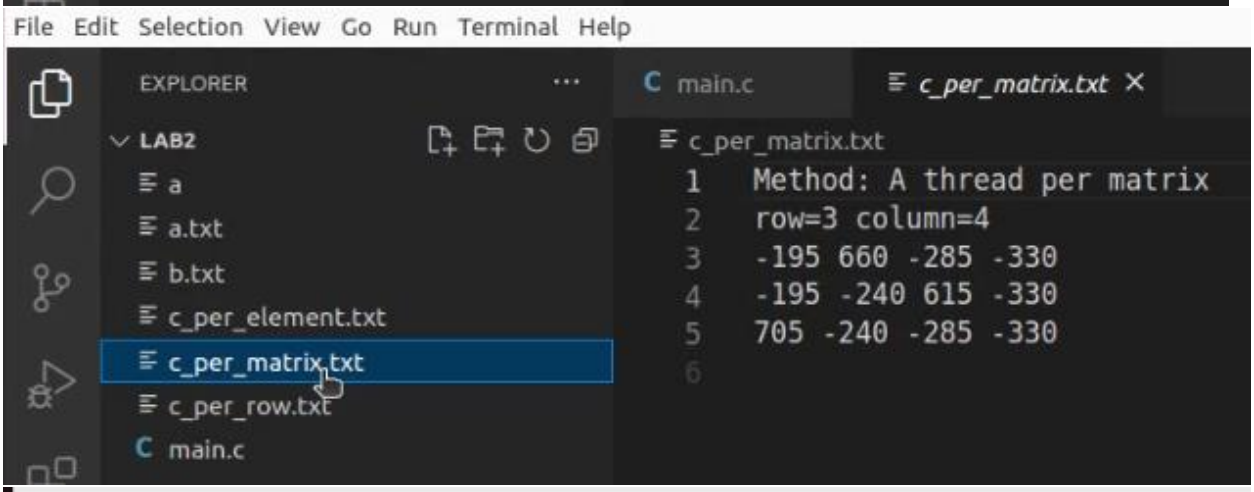

File Edit Selection View Go Run Terminal Help
EXPLORER
LAB2
a
a.txt
b.txt
c_per_element.txt
c_per_matrix.txt
c_per_row.txt
main.c
main.c
c_per_matrix.txt
c_per_matrix.txt
1 Method: A thread per matrix
2 row=10 column=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 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
13

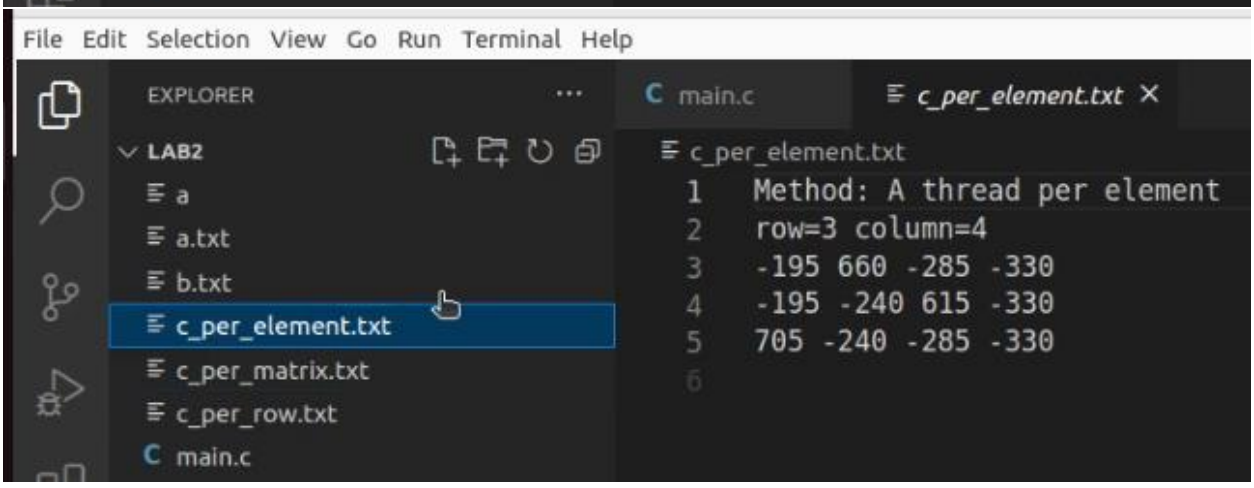

File Edit Selection View Go Run Terminal Help
EXPLORER
LAB2
a
a.txt
b.txt
c_per_element.txt
c_per_matrix.txt
c_per_row.txt
main.c
main.c
c_per_row.txt
c_per_row.txt
1 Method: A thread per row
2 row=10 column=10
3 415 430 445 460 475 490 505 520 535 550
4 940 980 1020 1060 1100 1140 1180 1220 1260 1300
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
13

Test 2


mahmoud@mahmoud-IdeaPad-Gaming-3-15IMH05:/media/mahmoud/Data/E/college/Term 8/Operating Systems/Lab2$
./a
////////////////Per Matrix Time////////////////
Number of threads created: 1
Seconds taken 0
Microseconds taken: 9
////////////////Per ROW Time////////////////
Number of threads created: 3
Seconds taken 0
Microseconds taken: 321
////////////////Per ELEMENT Time////////////////
Number of threads created: 12
Seconds taken 0
Microseconds taken: 572

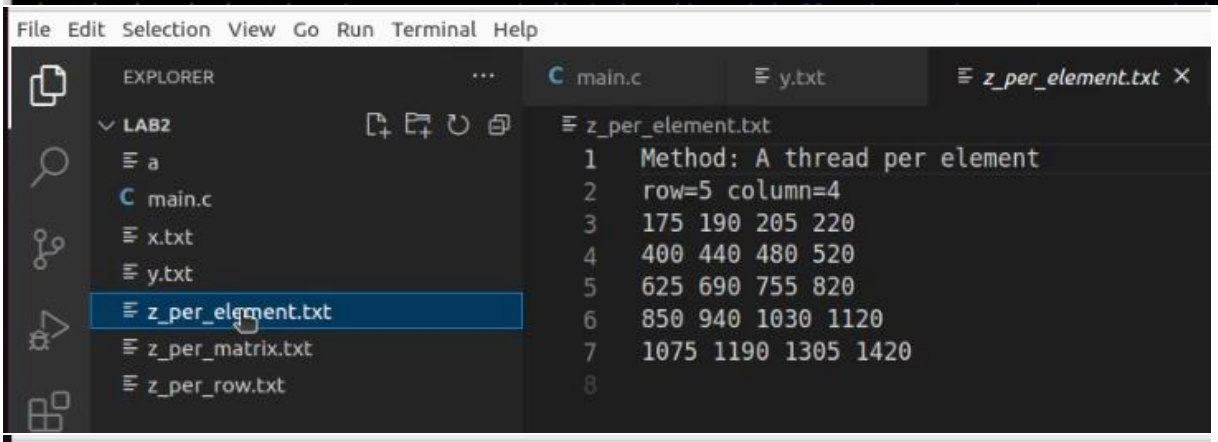

File Edit Selection View Go Run Terminal Help
EXPLORER
LAB2
a
a.txt
b.txt
c_per_element.txt
c_per_matrix.txt
c_per_row.txt
main.c
main.c
c_per_row.txt
c_per_row.txt
1 Method: A thread per row
2 row=3 column=4
3 -195 660 -285 -330
4 -195 -240 615 -330
5 705 -240 -285 -330
6

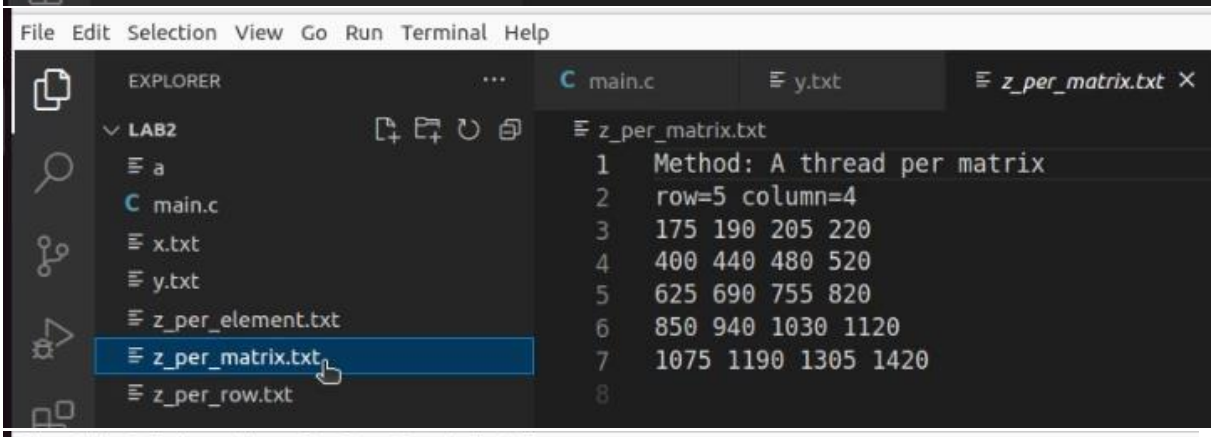

File Edit Selection View Go Run Terminal Help
EXPLORER
LAB2
a
a.txt
b.txt
c_per_element.txt
c_per_matrix.txt
c_per_row.txt
main.c
main.c
c_per_matrix.txt
c_per_matrix.txt
1 Method: A thread per matrix
2 row=3 column=4
3 -195 660 -285 -330
4 -195 -240 615 -330
5 705 -240 -285 -330
6

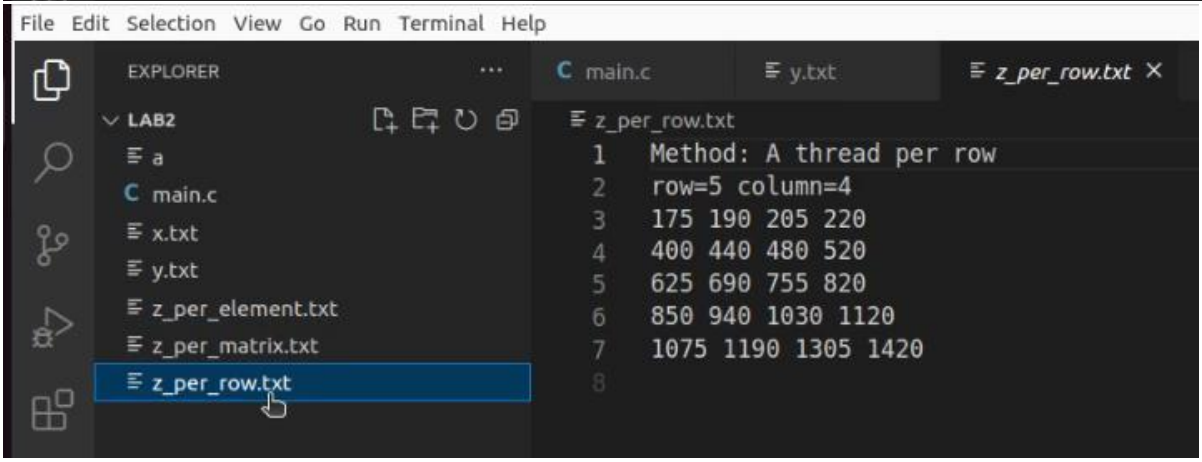

File Edit Selection View Go Run Terminal Help
EXPLORER
LAB2
a
a.txt
b.txt
c_per_element.txt
c_per_matrix.txt
c_per_row.txt
main.c
main.c
c_per_element.txt
c_per_element.txt
1 Method: A thread per element
2 row=3 column=4
3 -195 660 -285 -330
4 -195 -240 615 -330
5 705 -240 -285 -330
6

Test 3

```
mahmoud@mahmoud-IdeaPad-Gaming-3-15IMH05:/media/mahmoud/Data/E/college/Term 8/Operating Systems/Lab2$
 ./a x y z
////////////////Per Matrix Time////////////////
Number of threads created: 1
Seconds taken 0
Microseconds taken: 11
////////////////Per ROW Time////////////////
Number of threads created: 5
Seconds taken 0
Microseconds taken: 409
//////////////Per ELEMENT Time//////////////
Number of threads created: 20
Seconds taken 0
Microseconds taken: 285
```







Comparison

First method: multiplication is calculated using only the main thread.

Second method: for each row thread is created to calculate it so calculations is executed simultaneously, but there is the time for thread creation, so it won't be faster than method 1.

Third method: for each element thread is created to calculate it so calculations is executed simultaneously, but there is the time for thread creation, so it won't be faster than method 1.

Conclusion: multithreading is useful if we used it correctly if we create many threads without need that make the time increasing not decreasing.