Name: Ahmed Mohamed Fathallah. ID: 15.

Lab#2 Report

Code Organization:

- The program starts by preparing the file arguments passed to it.
- For each matrix file, if found a passed argument the file is opened. If file was not found "a.txt" or "b.txt" are opened to read input. Same goes for output either the argument or "c.out".
- Then the program reads matrix1 from file1 and matrix2 from file2.
- If number of columns of matrix1 doesn't equal number of rows of matrix2 the program halts indicating that multiplication can't be done.
- Method 1 starts creating a thread to handle each row of the resulting matrix3.
- A function called method 1 does the multiplication of each row of the output.
- After finishing method 1, method 2 starts by creating threads for each element of the output matrix.
- A function called method 2 does the multiplication of each element of the output.
- If errors are detected while creating threads the method indicates that an error happened and no output is presented.
- If no errors exist the method prints to the stdout the execution time and number of threads used, and the output matrix is printed to the output file.

Main Functions:

read_file1() & read_file2():

Each method reads a file and stores it into matrix1 and matrix2.

write_file():

Given the output file as a parameter it writes the output matrix to the file.

matrix_allocate():

responsible for the allocation of the matrices to the read dimensions from the files in order not to use any extra memory.

Method1():

The task done by threads created during method1. It takes the thread id representing the number of the row. Does the multiplication of the row of matrix1 to each column of matrix2 and stores results in matrix3.

Method2():

The task done by threads created during method2. It takes thread id and retrieves the row and column numbers from it by dividing and taking mod (m2) which is the number of columns of matrix2.

Does the multiplication of the row of matrix1 and column of matrix2 storing the output element in matrix3.

How to compile and run the code:

- Through the terminal, change the directory to the sent folder.
- Type make which will run the Makefile.
- An executable file named matMultp will be created.
- Invoke the file as in : ./matMultp [file1 file2 file3]
- The file names are optional.

Sample runs:

- Run 1

a × b × c_1 × c_2 ×

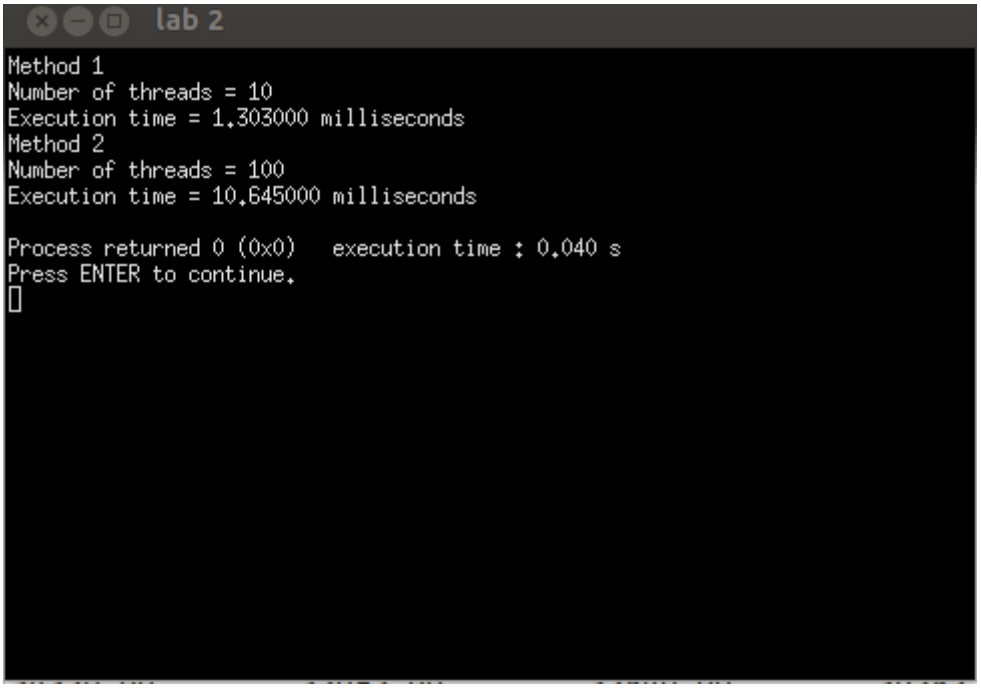
```
row=10 col=10
0       1       2       3       4       5       6       7       8       9
10      11      12      13      14      15      16      17      18      19
20      21      22      23      24      25      26      27      28      29
30      31      32      33      34      35      36      37      38      39
40      41      42      43      44      45      46      47      48      49
50      51      52      53      54      55      56      57      58      59
60      61      62      63      64      65      66      67      68      69
70      71      72      73      74      75      76      77      78      79
80      81      82      83      84      85      86      87      88      89
90      91      92      93      94      95      96      97      98      99
```

a × b × c_1 × c_2 ×

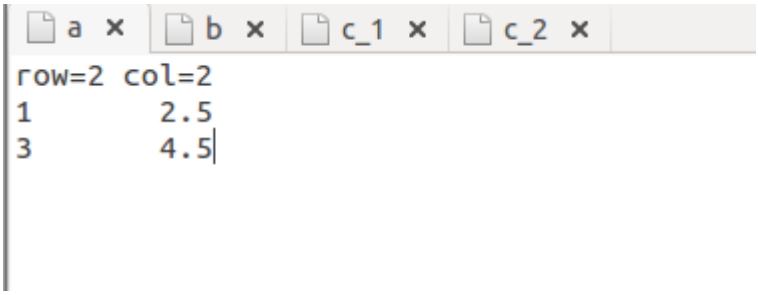
```
row=10 col=10
0       1       2       3       4       5       6       7       8       9
10      11      12      13      14      15      16      17      18      19
20      21      22      23      24      25      26      27      28      29
30      31      32      33      34      35      36      37      38      39
40      41      42      43      44      45      46      47      48      49
50      51      52      53      54      55      56      57      58      59
60      61      62      63      64      65      66      67      68      69
70      71      72      73      74      75      76      77      78      79
80      81      82      83      84      85      86      87      88      89
90      91      92      93      94      95      96      97      98      99
```

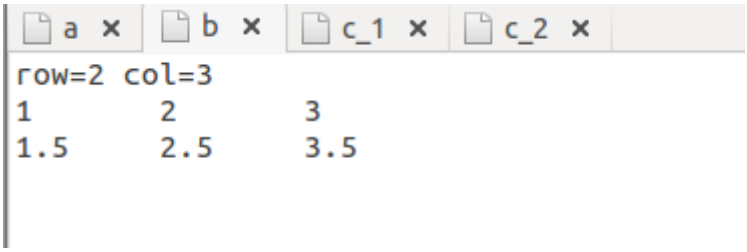
a × b × c_1 × c_2 ×

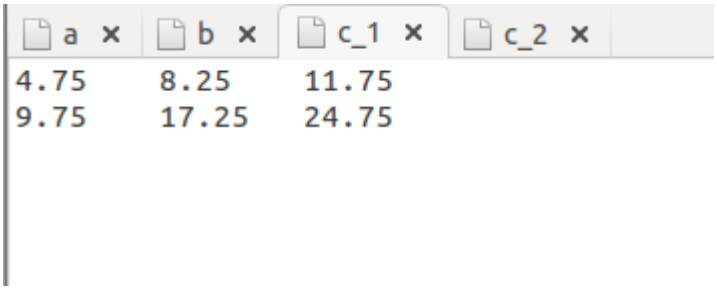
```
2850.00 2895.00 2940.00 2985.00 3030.00 3075.00 3120.00 3165.00 3210.00 3255.00
7350.00 7495.00 7640.00 7785.00 7930.00 8075.00 8220.00 8365.00 8510.00 8655.00
11850.00        12095.00        12340.00        12585.00        12830.00        13075.00        13320.00        13565.00
13810.00        14055.00
16350.00        16695.00        17040.00        17385.00        17730.00        18075.00        18420.00        18765.00
19110.00        19455.00
20850.00        21295.00        21740.00        22185.00        22630.00        23075.00        23520.00        23965.00
24410.00        24855.00
25350.00        25895.00        26440.00        26985.00        27530.00        28075.00        28620.00        29165.00
29710.00        30255.00
29850.00        30495.00        31140.00        31785.00        32430.00        33075.00        33720.00        34365.00
35010.00        35655.00
34350.00        35095.00        35840.00        36585.00        37330.00        38075.00        38820.00        39565.00
40310.00        41055.00
38850.00        39695.00        40540.00        41385.00        42230.00        43075.00        43920.00        44765.00
45610.00        46455.00
43350.00        44295.00        45240.00        46185.00        47130.00        48075.00        49020.00        49965.00
50910.00        51855.00
```

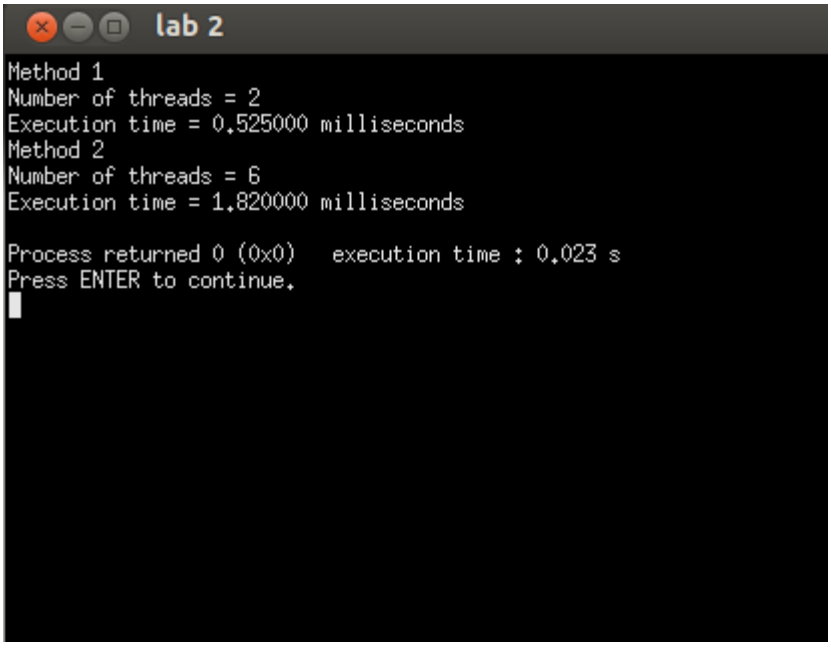

lab 2
Method 1
Number of threads = 10
Execution time = 1.303000 milliseconds
Method 2
Number of threads = 100
Execution time = 10.645000 milliseconds
Process returned 0 (0x0) execution time : 0.040 s
Press ENTER to continue.

- Run 2

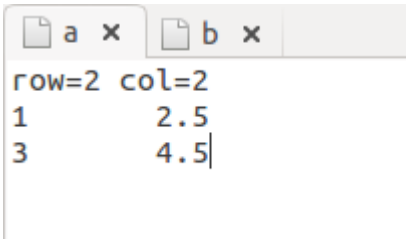

a
b
c_1
c_2
row=2 col=2
1 2.5
3 4.5

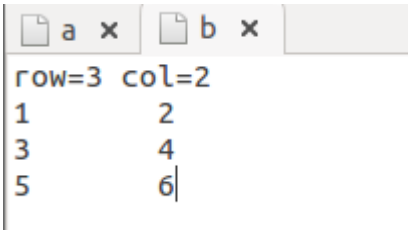

a
b
c_1
c_2
row=2 col=3
1 2 3
1.5 2.5 3.5

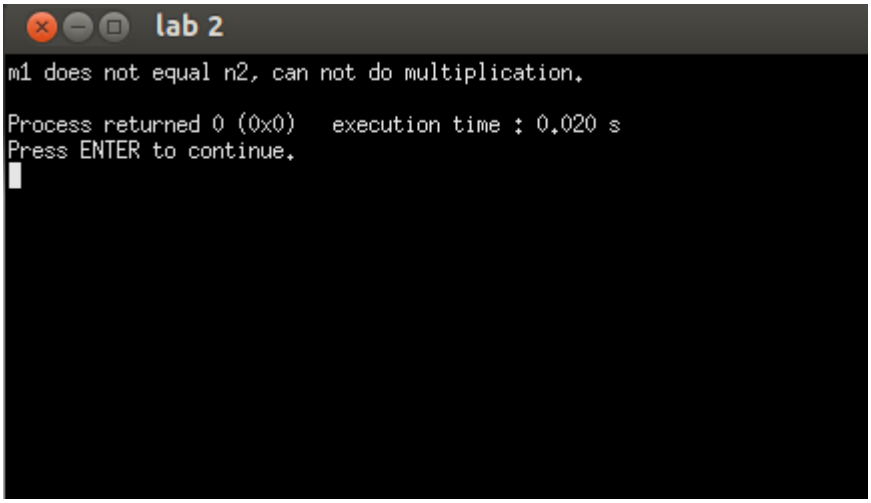

a
b
c_1
c_2
4.75 8.25 11.75
9.75 17.25 24.75


lab 2
Method 1
Number of threads = 2
Execution time = 0.525000 milliseconds
Method 2
Number of threads = 6
Execution time = 1.820000 milliseconds
Process returned 0 (0x0)   execution time : 0.023 s
Press ENTER to continue.

- Run 3 (error)


a
b
row=2 col=2
1 2.5
3 4.5


a
b
row=3 col=2
1 2
3 4
5 6


lab 2
m1 does not equal n2, can not do multiplication.
Process returned 0 (0x0)   execution time : 0.020 s
Press ENTER to continue.

Comparison between 2 methods:

Method 1:

Size	Threads number	Time (millisec)
5*5	5	0.849
10*10	10	1.924
50*50	50	9.384
100*100	100	36.77

Method 2:

Size	Threads number	Time (millisec)
5*5	25	3.566
10*10	100	13.331
50*50	2500	363.323
100*100	10000	1536.817

Clearly from the sample runs on different matrices dimensions, method1 is better than method2 regarding number of threads used and execution time.