

Faculty of Engineering  
Alexandria University  
CSED 2024  
Operating Systems

**OS**

***Lab 2 (matrix multiplication)***

***Bassant Yasser Salah 19017262***

# ***Matrix Multiplication (Multi-Threading):***

## **Problem Statement:**

Implement the multithreaded matrix multiplication using all three methods described above.

- Compare the three implementations according to the following:
  1. Number of threads created.
  2. Execution time taken.

## **i) How your code is organized:**

- 1- Taking the files of the input from the user in the terminal.
- 2- setting the files names using setdef();
  - in the setdef(): the values of the file will be read using the **read** method.
- 3-after setting up the values of the elements in the matrices, the execution method which contains the commands of each method call inside it will be called.
- 4- in each execution method: the thread of each way will be called to be executed and to print its time of execution.
  - inside the execution method there is **save()** to save the result of multiplication.

## **ii) Main functions:**

### ***Global Variables:***

```
char globalfile[1000]; //storing the name of the file in it.  
int solution[1000][1000];  
char argg[1000];  
//defining a new type to be used in the third method which is using a thread for  
each element.  
struct element{  
    int row;  
    int column;  
};
```

It is a method to store values of the matrices uploaded from a text file.

**Way:** this happens through looping over all elements in the text file and storing it using fscanf function(fptr,"%d",matrix[i][j]);

**upload\_valuesfiles(file name,name of the matrix that we will store all the values in it,row of the file, column):**

**Objective:** it is a method to store values of the matrices uploaded from a text file.

**Way:** this happens through looping over all elements in the text file and storing it using fscanf function(`fptr,"%d",matrix[i][j]`);

**read (path name,num which state which matrix we are working on right now):**

**Objective:** reading values from external text files and storing them using the previous mentioned method.

**Way:** the implementation happens as the following, first of all checking whether we are working on matrix1 or matrix 2 by checking num parameter.

After making sure of the given matrix, scanning the contents of the file and storing the values of row and column in the first line in the text file. After that, call the upload function to store the scanning elements from the file.

**Method1 (): //whole thread for the matrix**

**Objective:** multiplying the two matrices in a normal way (whole matrix).

**Way:** looping on each element of the first and second matrix to multiply them by each other and summing up the resulting multiplication to get the value of this process. `solution[i][j] += mat1[i][k]*mat2[k][j];`

**Method1\_thread (): //thread whole thread for the matrix**

**Objective:** creating a thread for the whole matrix.

**Way:** just create pthread\_t meth1 and thread it self using `pthread_create(meth1,NULL,method1(previous mentioned method),NULL);`

Second and third methods will be implemented in the same way but the only difference is in creating the threads and it's number as in method2 a thread will be created for each row after execution of multiplication of all row the threads will be closed using `pthread_join` function to avoid the sequential method execution. While in method 3, a thread will be created for each element and of course after

the execution of multiplication and finishing the execution of all elements threads they will terminate using `p_thread join`.

### **printing (matrix,row,col):**

**Objective:** printing out the elements of the matrix.

**Way:** looping on each element of any given matrix and printing it.

### **saving (path):**

**Objective:** saving the result of the multiplication in an external text file and its name will be given from the user or default in case of not providing it.

**Way:** creating a file and opening it using `fopen(path,"w"`=the state of the file which is written in it). Looping through each element of the solution matrix and writing in the file using `fprintf(predefined created file (file),element itself)`.

### **excution1 ():**

**Objective:** method for running the method1 and printing out its execution time.

**Way:** first of all, we will start the timer for the execution then will call the `method1_thread` to be executed and save the resultant of the method in an external file using the saving method. After finishing the execution of the method we will stop using `gettimeofday(&stop, NULL)` the timer and print out the difference between start and end of execution. And

Execution 3 and execution 2 are implemented in the same way but the only difference is calling the `thread_method3()` and `thread_method3()` respectively.

### **setdef (int set(integer that state the shape of the input either there is an argument or not),argv):**

**Objective:**taking the input from the user for the required file tests.

**Way:** according to the number of the set, the method will execute lines. First of all, we have defined previously a global string variable named `globalfile` which is used to store the name of the text file after formatting it or adding the extension ".txt". Empty this global variable and concat in it the given name of the file if given by the user or set to default in case the user didn't provide the name. We will do

the same process for each file name and according to the given case. At the end after defining the name of the files we will upload its values and read them from the file using the **read method**.

**main (int argc(which count the number of the given commands),char \*argv[]storing the arguments given from the user):**

**Objective:** calling the function of the setdef to return the values in the matrices then the multiplication execution method will be called to perform multiplication and return the result in the text file.

### **ii) How to run and compile the code:**

1- Download the main files and put them in separate files from other folders.

2-inside the created file that contains main.c and tests folders, open the terminal either by choosing terminal from list appeared after right click in any place in the folder or by pressing on **CTRL+ALT+t** at the same time.

3- After opening the terminal, we will run the code in the file using **make which will compile the code directly**.

4- in case you wanna test a file without an argument just write **./matmultp** and the files have to be in the same directory as the matmultp.c.

5- In case of test cases with arguments, write the following command **./main test1/a test1/b test1/c** (test1 is the name of the folder ,a b are the names of the text inputs while c is the output file).

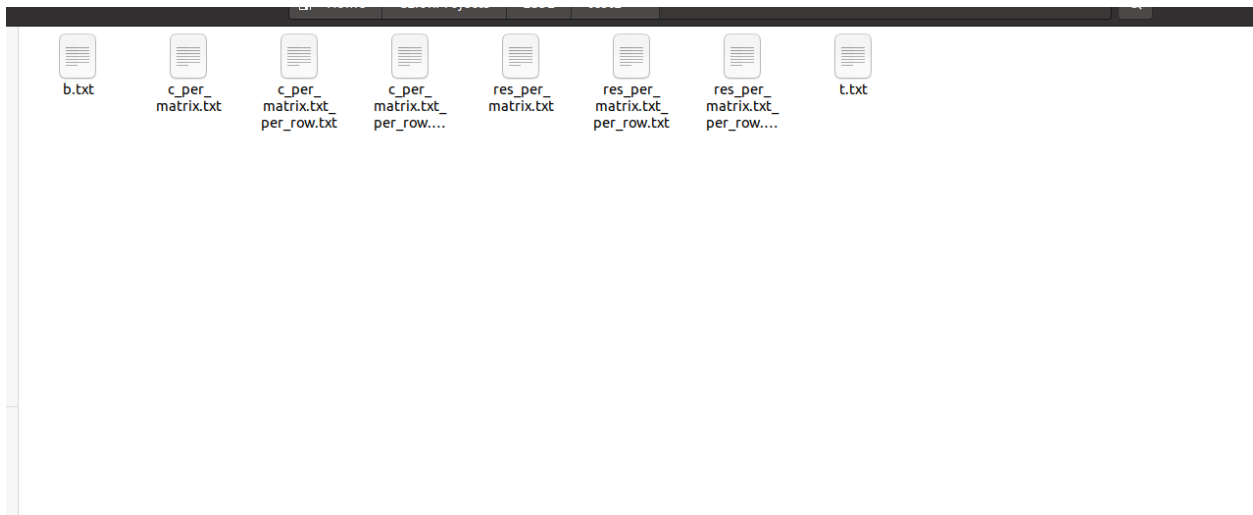
#### iv) Sample runs:

*test1*

```
bassant@bassant-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/CLionProjects/Lab2$ ./matMultp test1/a test1/b test1/c
Method 1
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
Number of Threads used: 1
Seconds taken 0
Microseconds taken: 122
-----
Method 2
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
Number of Threads used: 10
Seconds taken 0
Microseconds taken: 395
-----
Method 3
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
Number of Threads used: 100
Seconds taken 0
Microseconds taken: 1514
-----
```

## Test2

```
bassant@bassant-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/CLionProjects/Lab2$ ./matMultp test2/t test2/b test2/res
Method 1
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
Number of Threads used: 1
Seconds taken 0
Microseconds taken: 84
-----
Method 2
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
Number of Threads used: 3
Seconds taken 0
Microseconds taken: 429
-----
Method 3
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
Number of Threads used: 12
Seconds taken 0
Microseconds taken: 400
-----
```

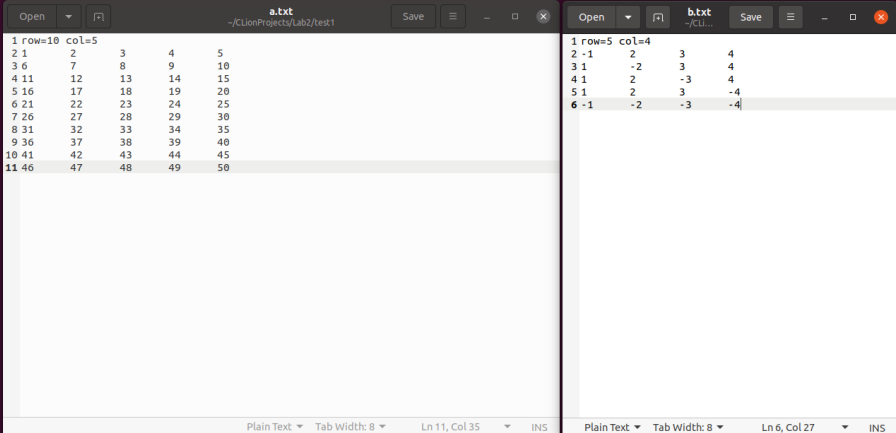


## Test3

```
-----
bassant@bassant-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/CLionProjects/Lab2$ ./matMultp test3/a test3/b test3/res
WARNING!!! not valid Dimensions
```

# Test4

```
bassant@bassant-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/CLionProjects/Lab2$ ./matMult test1/a test2/b test1/res1
Method 1
3 2 -3 -12
8 12 12 8
13 22 27 28
18 32 42 48
23 42 57 68
28 52 72 88
33 62 87 108
38 72 102 128
43 82 117 148
48 92 132 168
Number of Threads used: 1
Seconds taken 0
Microseconds taken: 233
-----
Method 2
3 2 -3 -12
8 12 12 8
13 22 27 28
18 32 42 48
23 42 57 68
28 52 72 88
33 62 87 108
38 72 102 128
43 82 117 148
48 92 132 168
Number of Threads used: 10
Seconds taken 0
Microseconds taken: 1059
-----
Method 3
3 2 -3 -12
8 12 12 8
13 22 27 28
18 32 42 48
23 42 57 68
28 52 72 88
33 62 87 108
38 72 102 128
43 82 117 148
48 92 132 168
Number of Threads used: 40
Seconds taken 0
Microseconds taken: 2297
-----
```

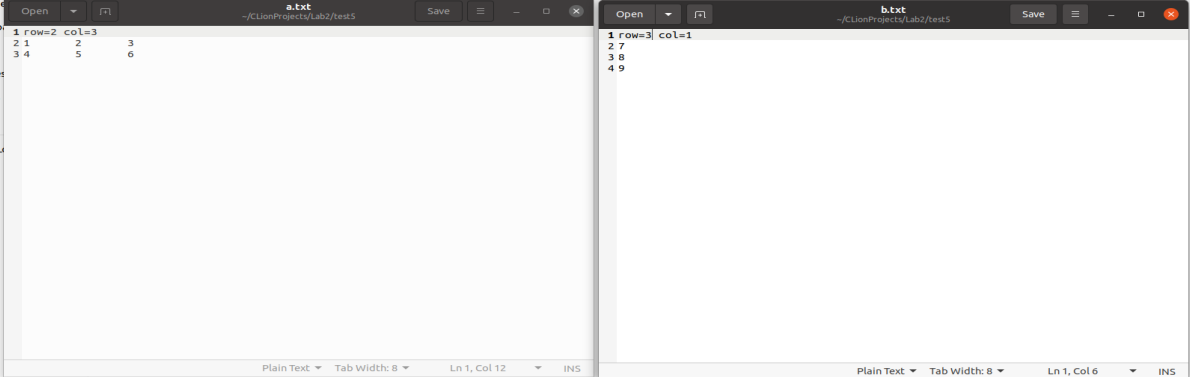


2	1	2	3	4
3	6	7	8	9
4	11	12	13	14
5	16	17	18	19
6	21	22	23	24
7	26	27	28	29
8	31	32	33	34
9	36	37	38	39
10	41	42	43	44
11	46	47	48	49

2	-1	2	3
3	1	-2	3
4	1	2	-3
5	1	2	3
6	-1	-2	-3

# Test5

```
bassant@bassant-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/CLionProjects/Lab2$ ./matMult test5/a test5/b test5/res
Method 1
50
122
Number of Threads used: 1
Seconds taken 0
Microseconds taken: 230
-----
Method 2
50
122
Number of Threads used: 2
Seconds taken 0
Microseconds taken: 707
-----
Method 3
50
122
Number of Threads used: 2
Seconds taken 0
Microseconds taken: 462
-----
```



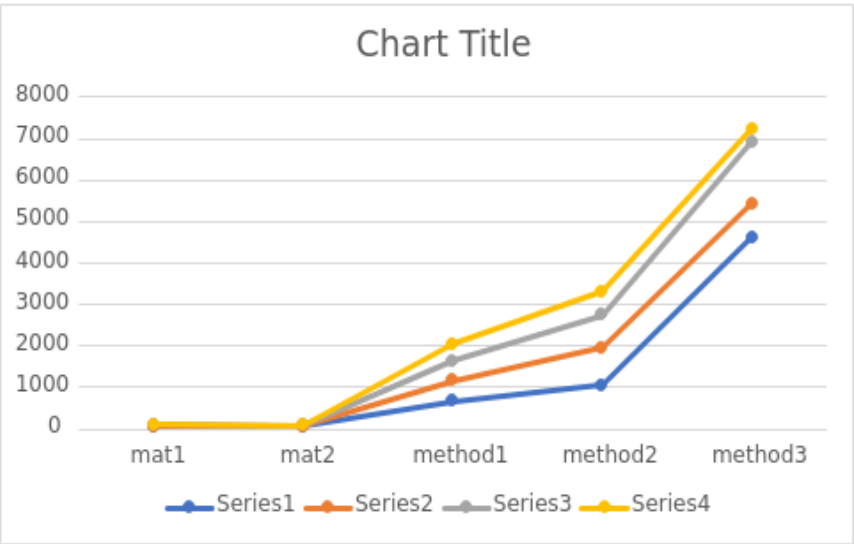
2	1	2
3	4	5
4	5	6

2
3
4



v)A comparison between the three methods of matrix multiplication.

From what we have seen in the test cases and the time printed in different cases, we have found that in small number of elements of matrices method 2 and method 3 near to each other in time analysis while method 1 is better than both of them but by increasing the number of elements, the execution time of method 3 became worse than method 2 which means that it is the more time consuming one among 3 ways while method 1 is the best and fastest.



25	mat1	mat2	method1	method2	method3	
26	50	50	677	1052	4625	
27	20	15	482	897	807	
28	25	16	486	796	1491	
29	6	3	404	572	321	
30						