# Lab 2 : Threads

Name : Louay Magdy AbdelHalim Ali

ID : 19016195

# Code Organization & main functions description

## 1st: variables

most of variables will be found global such as the three matrices A, B, and C. their dimesions together with pointers to the files from which they are read / written.

You will also find a user defined struct to hold the row and column index in case of thread per element

## 2nd: File-Reading & tokenization functions

*file-reading* are the first functions to be executed :

- there is a function called *create files* that create files that we will write to it but the the input files as a.txt and b.txt have to be existed
- there is a function called *read files* that read contents of the input files and classify input according to :
  - ✔ which matrix is addressed
  - ✔ whether the line read is the first line that contain dimensions where split goes over spaces then = sign or it is any other line where we can split the line over spaces or tabs to get the entries
- there is a function called *create Custom files* that deals with main function *argv* parameter to get the name of custom files

note : the user must create the custom files or a.txt and b.txt in case they are not found

## 3rd: first-type thread functions

there are 2 functions :

- *thread per matrix* function to create the single thread responsible for the operation and make sure that the matrices can be multiplied
- *multiply matrix by matrix* function that multiplies the 2 matrices together and write the result into files as well → thread function

### 4th: second-type thread functions

there are 2 functions :

- *thread per row* function that make sure that matrices can be multiplied and if so, it creates a thread for each row in matrix A
- *multiply row by matrix* function that multiplies a row of given index with matrix B as a whole → thread function

### 5th: third-type thread functions

there are also 2 functions :

- *thread per element* function that make sure that functions can be multiplied and creates a thread for ever row in A and col. in B
- *multiply row by col* function that multiplies a given row in A with a given col. in B to get an entry in C and notice that all the function requirements are set before the loop of creating threads so as not to introduce any overhead during thread creation → thread function

## How to compile and run the code

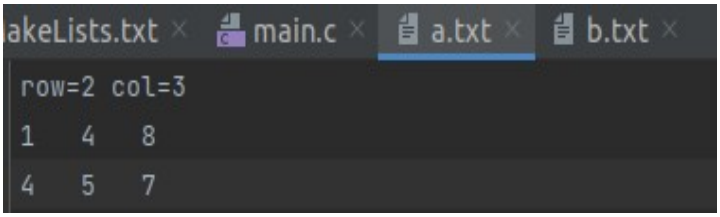
- Open the terminal and change the directory till you get in the directory where the project is *main.c* file of project is located
- run the following commands : *gcc main.c -o matMultp -pthread*
- now the code is compiled. change your directory to : *cmake-build-debug*
- open the the executable file *matMultp* found there using : *./matMultp* or *./matMultp args.* In case there are custom input files but make sure you have created them In the project file

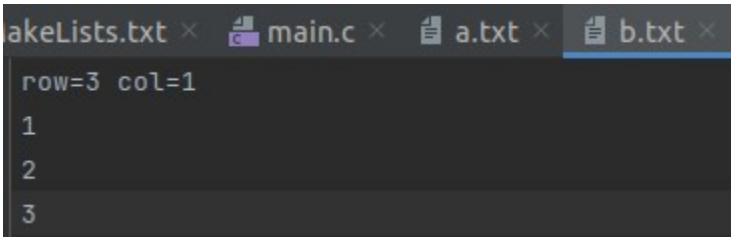
### you can also compile it using:

- command *make* In the project directory
- running the executable *matMultp1* found in *cmake-build-debug* directory

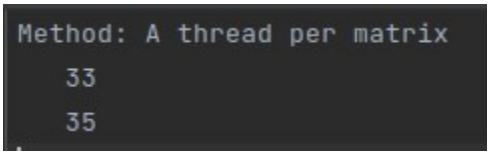
# Sample Cases:

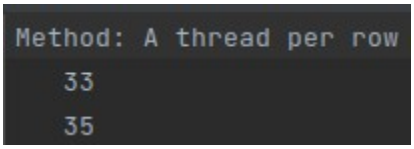
1- testing prog

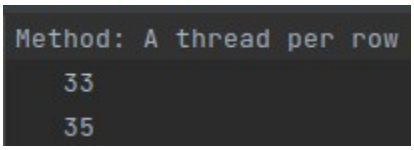


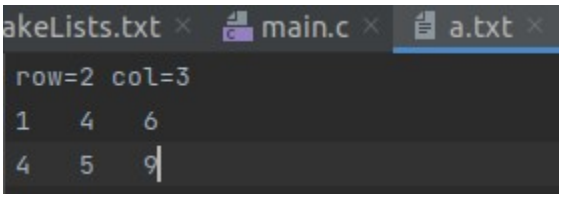

result

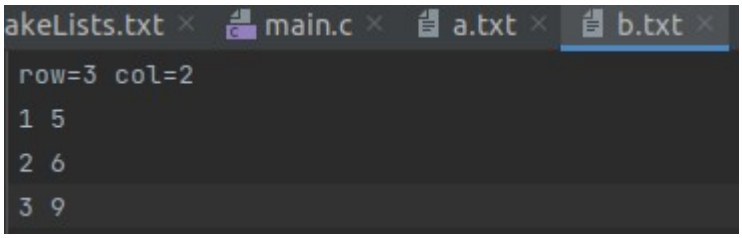


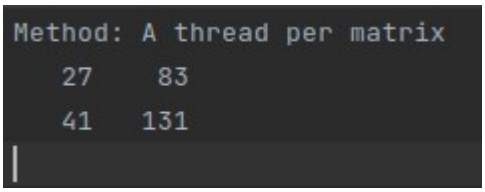



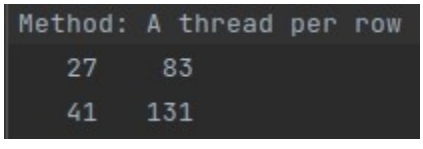

2- testing prog.

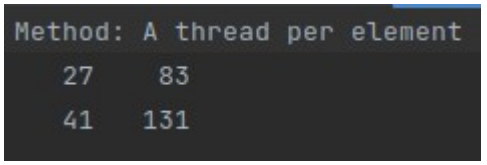



result:



3 – testing from the executable file and seeing the result in the console

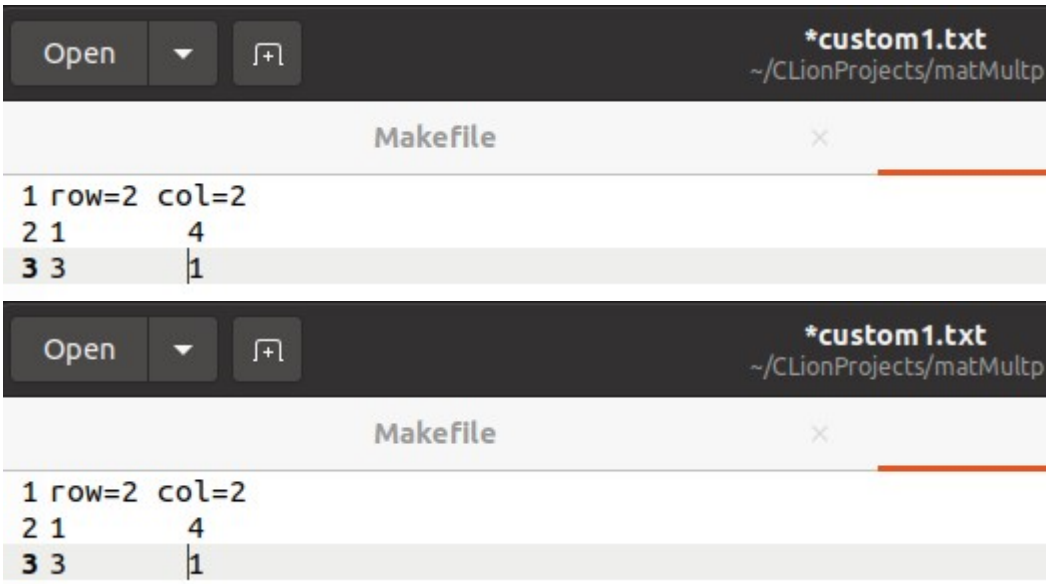
also the files have these values printed

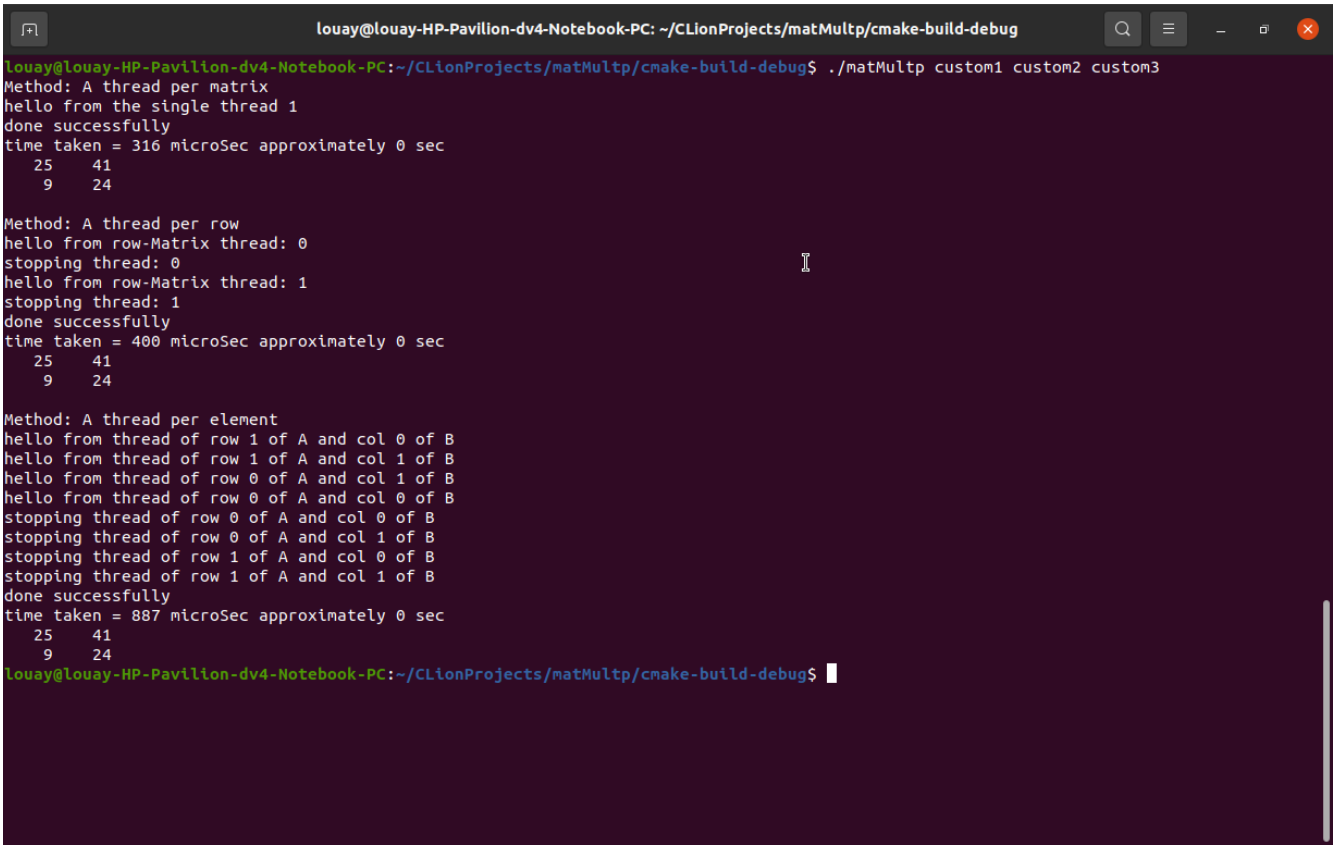
```
louay@louay-HP-Pavilion-dv4-Notebook-PC:~/CLionProjects/matMultp/cmake-build-debug$ ./matMultp
Method: A thread per matrix
hello from the single thread 1
done successfully
time taken = 413 microSec approximately 0 sec
   27    83
   41   131

Method: A thread per row
hello from row-Matrix thread: 1
hello from row-Matrix thread: 0
stopping thread: 0
stopping thread: 1
done successfully
time taken = 638 microSec approximately 0 sec
   27    83
   41   131

Method: A thread per element
hello from thread of row 0 of A and col 0 of B
hello from thread of row 0 of A and col 1 of B
stopping thread of row 0 of A and col 0 of B
stopping thread of row 0 of A and col 1 of B
hello from thread of row 1 of A and col 1 of B
hello from thread of row 1 of A and col 0 of B
stopping thread of row 1 of A and col 0 of B
stopping thread of row 1 of A and col 1 of B
done successfully
time taken = 411 microSec approximately 0 sec
   27    83
   41   131
```

4 – testing using custom arguments






seeing one of the resulting files custom3_per_element.txt

```
Method: A thread per element
   25    41
    9    24
```

# Comparison between the three methods according to time taken :

running various cases :

```
louay@louay-HP-Pavilion-dv4-Notebook-PC:~/CLionProjects/matMultp/cmake-build-debug$ ./matMultp custom1 custom2 custom3
Method: A thread per matrix
hello from the single thread 1
done successfully
time taken = 350 microSec approximately 0 sec
   25    41
    9    24

Method: A thread per row
hello from row-Matrix thread: 1
hello from row-Matrix thread: 0
stopping thread: 0
stopping thread: 1
done successfully
time taken = 443 microSec approximately 0 sec
   25    41
    9    24

Method: A thread per element
hello from thread of row 0 of A and col 1 of B
hello from thread of row 0 of A and col 0 of B
stopping thread of row 0 of A and col 0 of B
stopping thread of row 0 of A and col 1 of B
hello from thread of row 1 of A and col 0 of B
stopping thread of row 1 of A and col 0 of B
hello from thread of row 1 of A and col 1 of B
stopping thread of row 1 of A and col 1 of B
done successfully
time taken = 1033 microSec approximately 0 sec
   25    41
    9    24
```

```
louay@louay-HP-Pavilion-dv4-Notebook-PC:~/CLionProjects/matMultp/cmake-build-debug$ ./matMultp
Method: A thread per matrix
hello from the single thread 1
done successfully
time taken = 303 microSec approximately 0 sec
   27    83
   41   131

Method: A thread per row
hello from row-Matrix thread: 0
stopping thread: 0
hello from row-Matrix thread: 1
stopping thread: 1
done successfully
time taken = 718 microSec approximately 0 sec
   27    83
   41   131

Method: A thread per element
hello from thread of row 1 of A and col 0 of B
hello from thread of row 0 of A and col 1 of B
hello from thread of row 0 of A and col 0 of B
stopping thread of row 0 of A and col 0 of B
stopping thread of row 0 of A and col 1 of B
stopping thread of row 1 of A and col 0 of B
hello from thread of row 1 of A and col 1 of B
stopping thread of row 1 of A and col 1 of B
done successfully
time taken = 1293 microSec approximately 0 sec
   27    83
   41   131
```

```
time taken = 385 microSec approximately 0 sec
    9    29
   14    50

Method: A thread per row
hello from row-Matrix thread: 1
hello from row-Matrix thread: 0
stopping thread: 0
stopping thread: 1
done successfully
time taken = 756 microSec approximately 0 sec
    9    29
   14    50

Method: A thread per element
hello from thread of row 0 of A and col 1 of B
hello from thread of row 0 of A and col 0 of B
stopping thread of row 0 of A and col 0 of B
stopping thread of row 0 of A and col 1 of B
hello from thread of row 1 of A and col 0 of B
stopping thread of row 1 of A and col 0 of B
hello from thread of row 1 of A and col 1 of B
stopping thread of row 1 of A and col 1 of B
done successfully
time taken = 790 microSec approximately 0 sec
```

it seems in normal cases, that the single threaded application is the fastest at all although of its high time complexity $O(n^3)$ but the thread creation/join overhead which is large is very small compared to the other two function where there are many threads

it seems also that thread per row function will be faster than thread per element because the time complexity of each thread function in the thread per row function is more than that in thread in element (get rid of high complexity in parallel). Also the overhead is less in case of thread-per-row function