Threads
OS lab 2

name :mina sameh labib
id :75

Contents

Code Organization

the program consists of 2 methods for calculating matrix multiplication:

- method 1

  we run a thread for each row inside that thread we multiply the row from first matrix with each column in the other matrix to get a row of the output matrix.

```
method 1:
for(i=0;i<Crows;i++) //call thread 1

thread 1:
for(j=0;j<Bcolumns;j++){
    double res=0;
        for(i=0;i<Acolumns;i++)
            res += A[rowID][i]*B[i][j];
    C[rowID][j] = res;
}
```

- method 2

  we run a thread for each cell inside that thread we multiply a row from the first matrix with a column in the other matrix to get a cell of the output matrix.

```
method 2:
for(i=0;i<Crows;i++)
      for(j=0;j<Ccolumns;j++)
            //call thread 2

 thread 2:
    double res=0;
    for(i=0;i<Acolumns;i++)
        res += A[rowID][i]*B[i][ColumnID];
    C[rowID][ColumnID]=res;
```

Main Functions

- int validFiles()

  returns 0 if one of the files can’t be accessed and 1 otherwise.
- void readFiles()

  reads the matrices from files specified by user or the default files (a.txt, b.txt) then store the two matrices in double 2D array for next operation.
- void *rowCalc(void *row)

  given a row id this function calculates its values from that row in the first matrix multiplied by each column of the other matrix
- void *cellCalc(void *req)

  given a row and column id this function calculates that cell from a row in the first matrix multiplied by a column of the other matrix
- void method1()

  run the first method
- void method2()

  run the second method
- void writeCtoFile(int method)

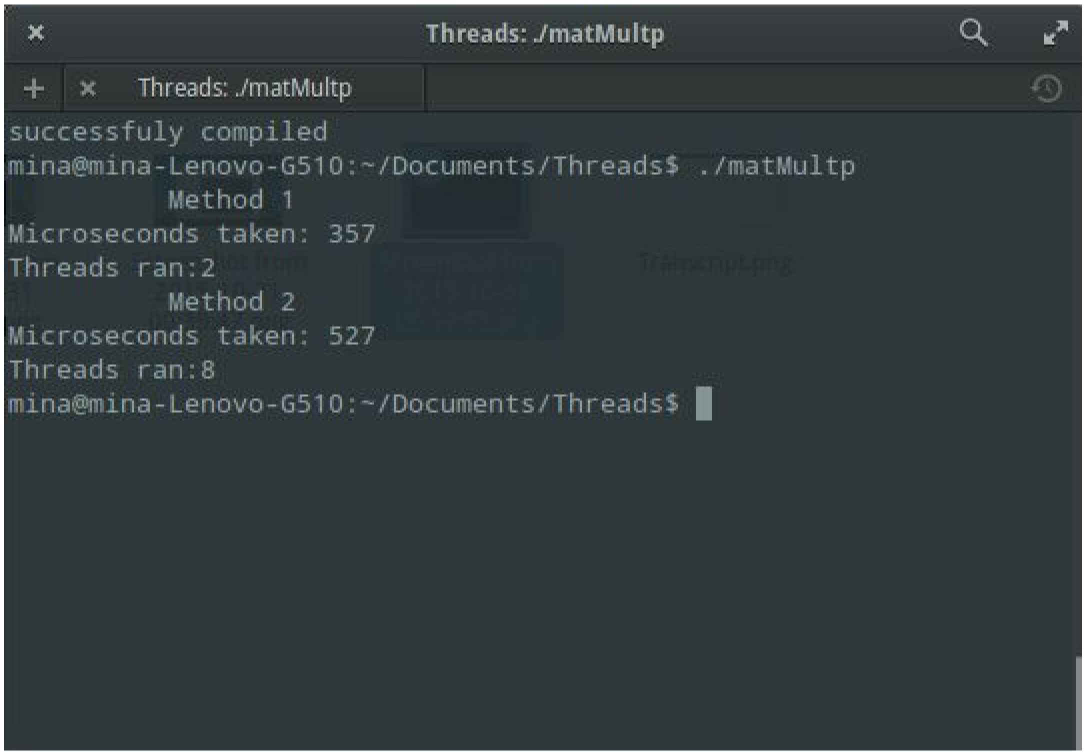
  writes the output matrix to a file specified by user or default file c_[method].out, it work with the two cases output file has extension or not, if it has extension the output file will be
  [fileName]_[method].[extension]
  if it has no extension the output file will be [fileName]_[method]

How To Compile the Project

1. start terminal
2. change directory to the folder containing the source code and the makefile
3. run make command in terminal
4. you can run the shell by typing ./matMultp Mat1 Mat2 MatOut

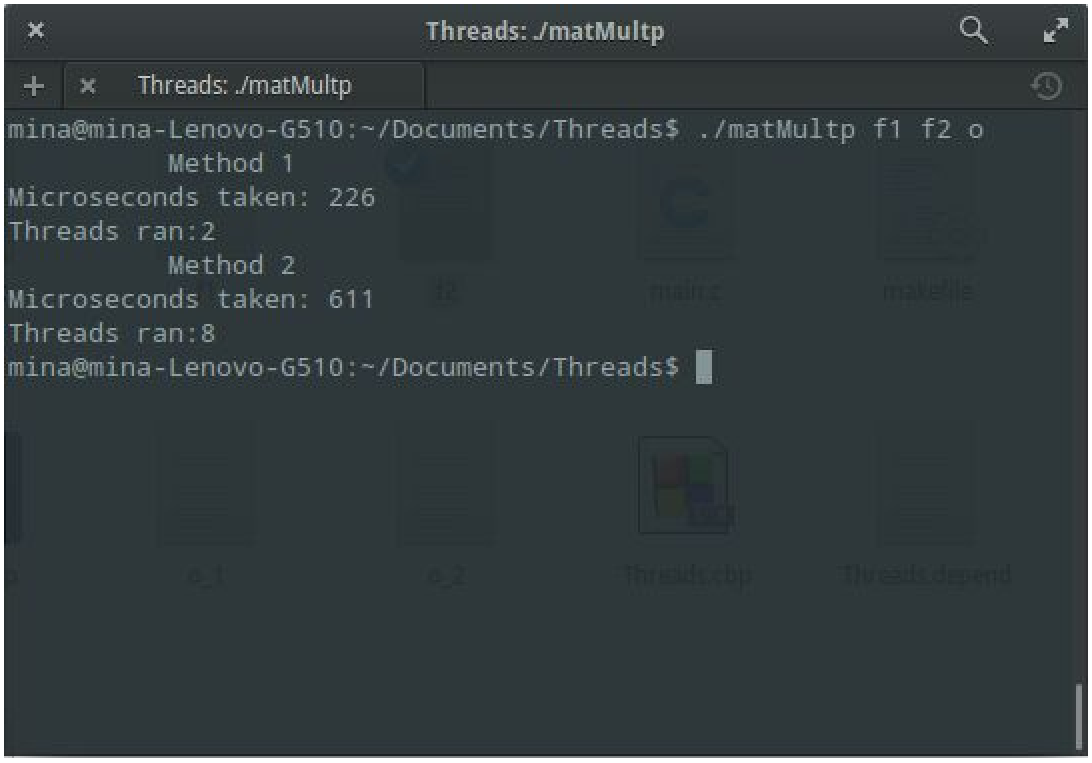
Sample runs

- default case input files a.txt, b.txt output files c_1.out, c_2.out



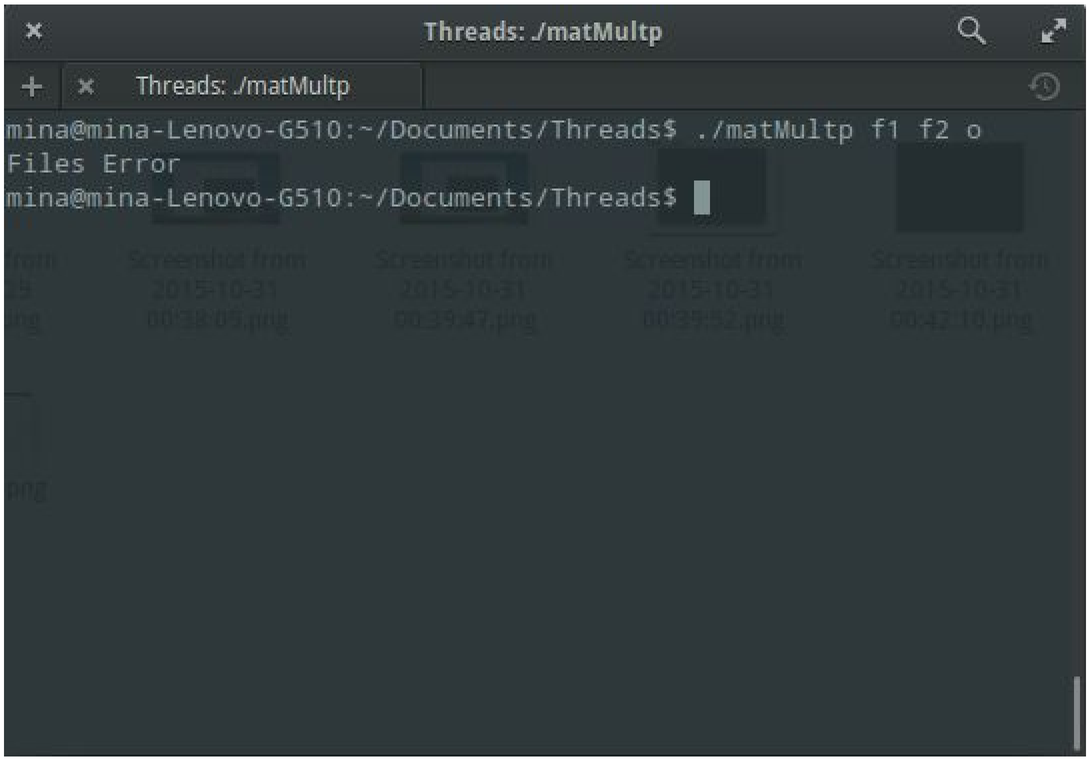
```
successfuly compiled
mina@mina-Lenovo-G510:~/Documents/Threads$ ./matMultp
         Method 1
Microseconds taken: 357
Threads ran:2
         Method 2
Microseconds taken: 527
Threads ran:8
mina@mina-Lenovo-G510:~/Documents/Threads$
```

- user specify files



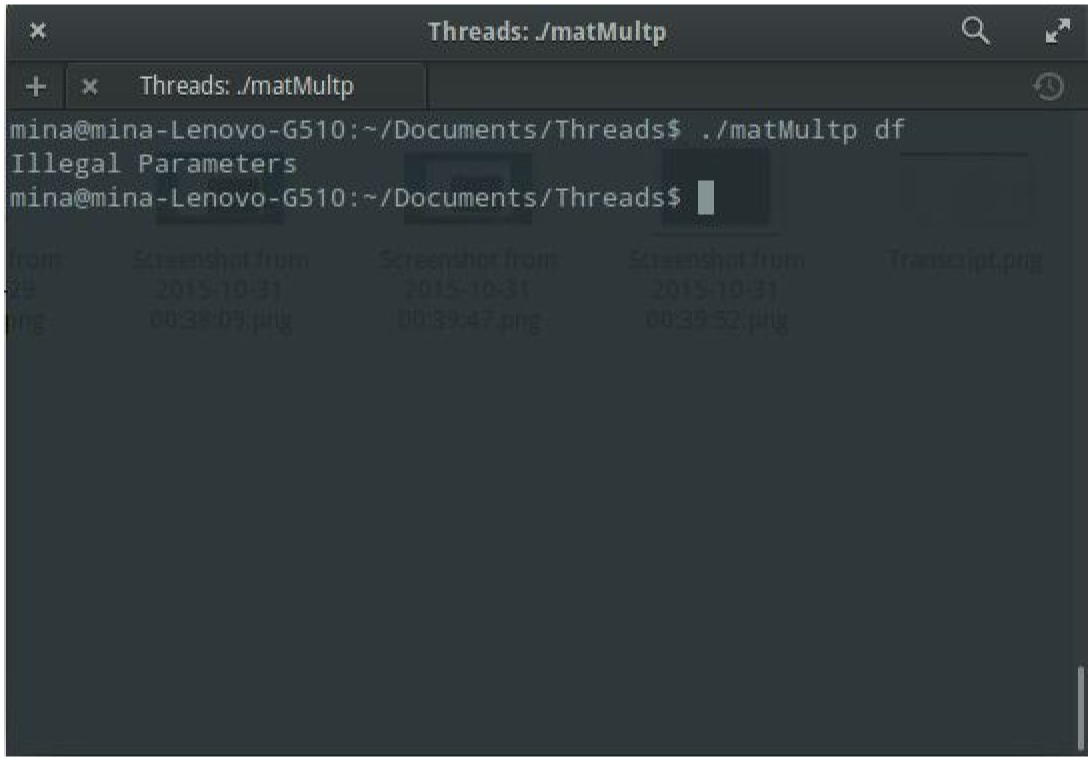
```
mina@mina-Lenovo-G510:~/Documents/Threads$ ./matMultp f1 f2 o
          Method 1
Microseconds taken: 226
Threads ran:2
          Method 2
Microseconds taken: 611
Threads ran:8
mina@mina-Lenovo-G510:~/Documents/Threads$
```

- specified files not found



```
mina@mina-Lenovo-G510:~/Documents/Threads$ ./matMultp f1 f2 o
Files Error
mina@mina-Lenovo-G510:~/Documents/Threads$
```

- user gives wrong parameters



```
mina@mina-Lenovo-G510:~/Documents/Threads$ ./matMultp df
Illegal Parameters
mina@mina-Lenovo-G510:~/Documents/Threads$
```

Comparison between method1 & method2

let’s assume A_{mXn} X B_{nXk} = C_{mXk}

- method 1 will create m threads each will calculate a row
- method 2 will create m*k threads each will calculate a cell

creating a thread in method 2 will take more time than its work (calculating a cell) so it will be slow and method 1 will be better to use.

time in usec	2x2 X 2x4	4x5 X 5x10	5x10 X 10x15	150x150 X 150x150
method 1	226	734	421	10164
method 2	611	2157	3222	426217