Lap2

Name: Mohamed Ahmed Aly Riyad

Id: 20011457

Problem statement:

You are required to implement a multi-threaded matrix multiplication program.
The input to the program is two matrixes A(x*y) and B(y*z) that are read from corresponding text files. The output is a matrix C(x*z) that is written to an output text file.
A parallelized version of matrix multiplication can be done using one of these three methods:

1. A thread computes the output C matrix i.e. without multi-threading. (A thread per matrix).
2. A thread computes each row in the output C matrix. (A thread per row).
3. A thread computes each element in the output C matrix. (A thread per element).

A description of the overall organization of your code

1.taking input as arguments for main.

2.open files for input and create files for output .

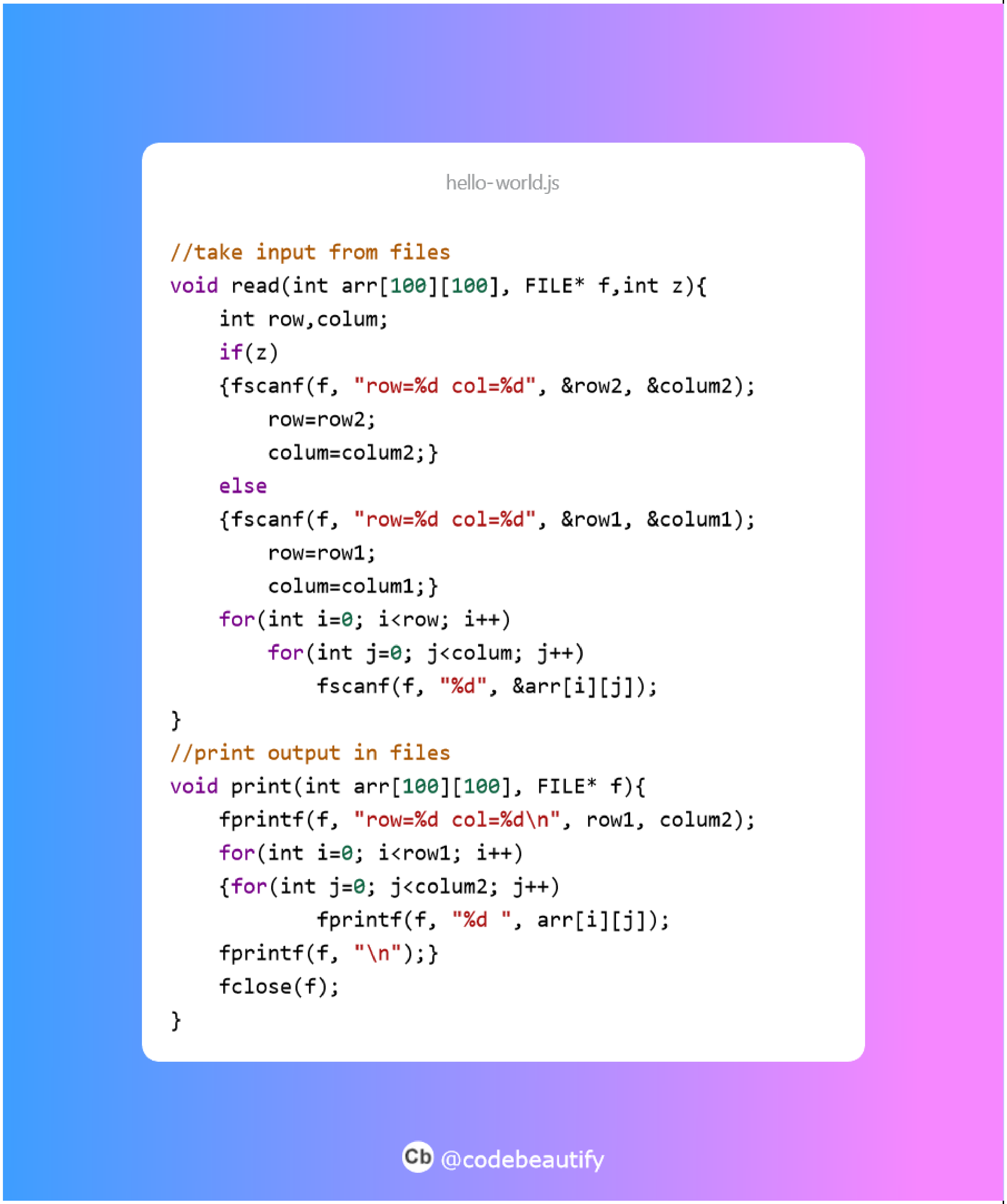
3. 3 methods for Matrix Multiplication.

4.creation threads for every method.

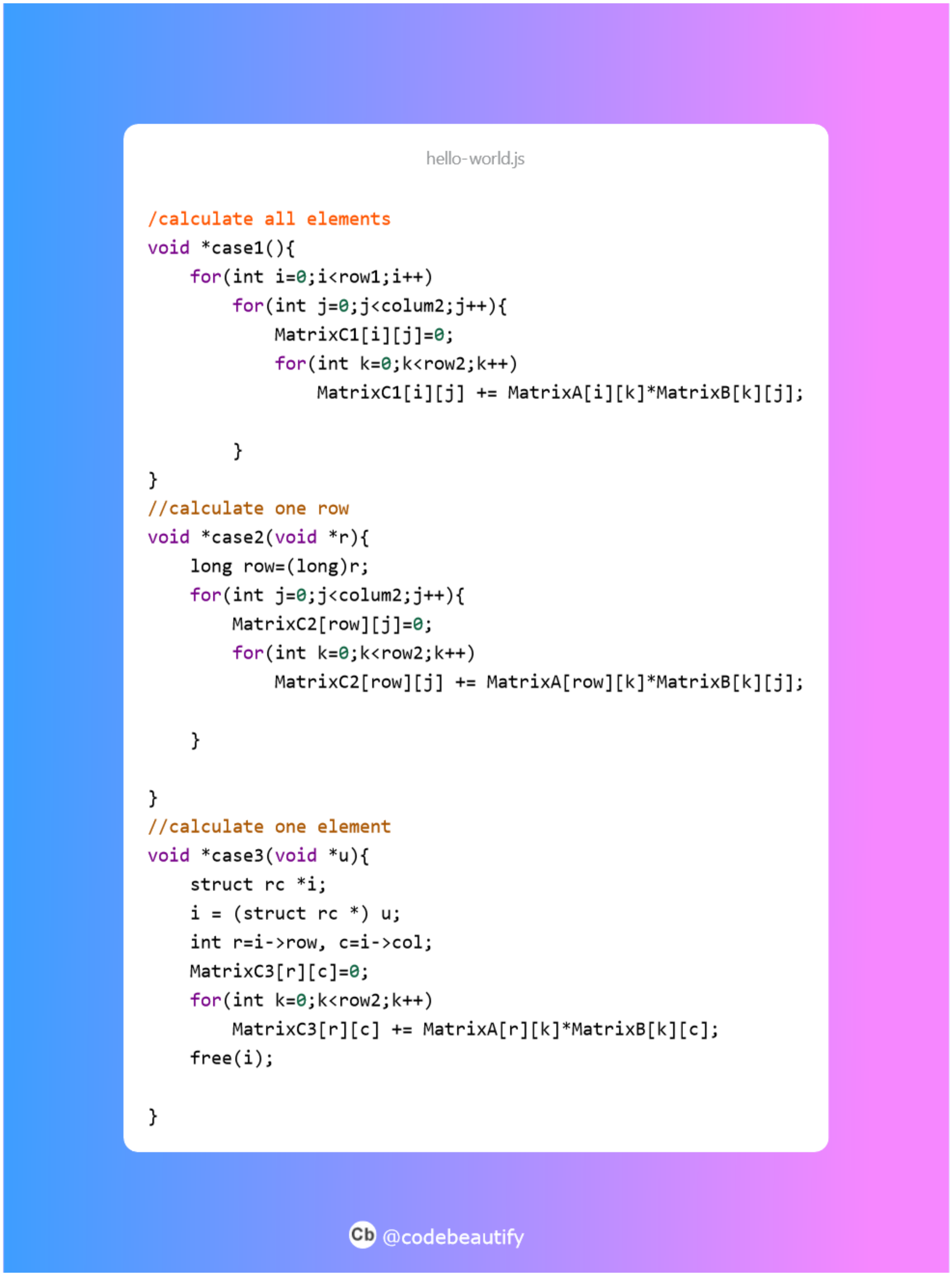
5.calculate time for each method and print it.

6.calculate number of threads of each methods.

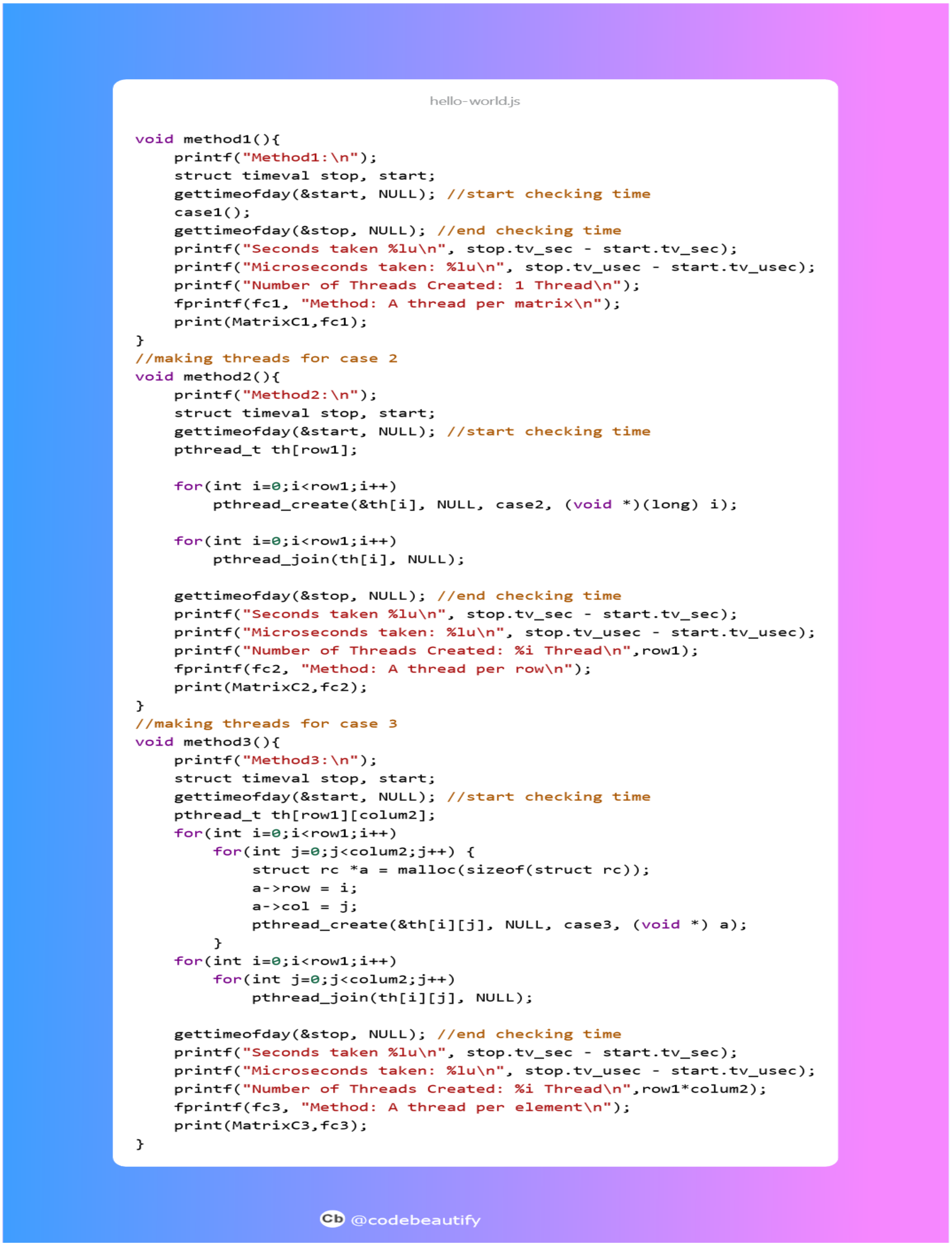
The major function:



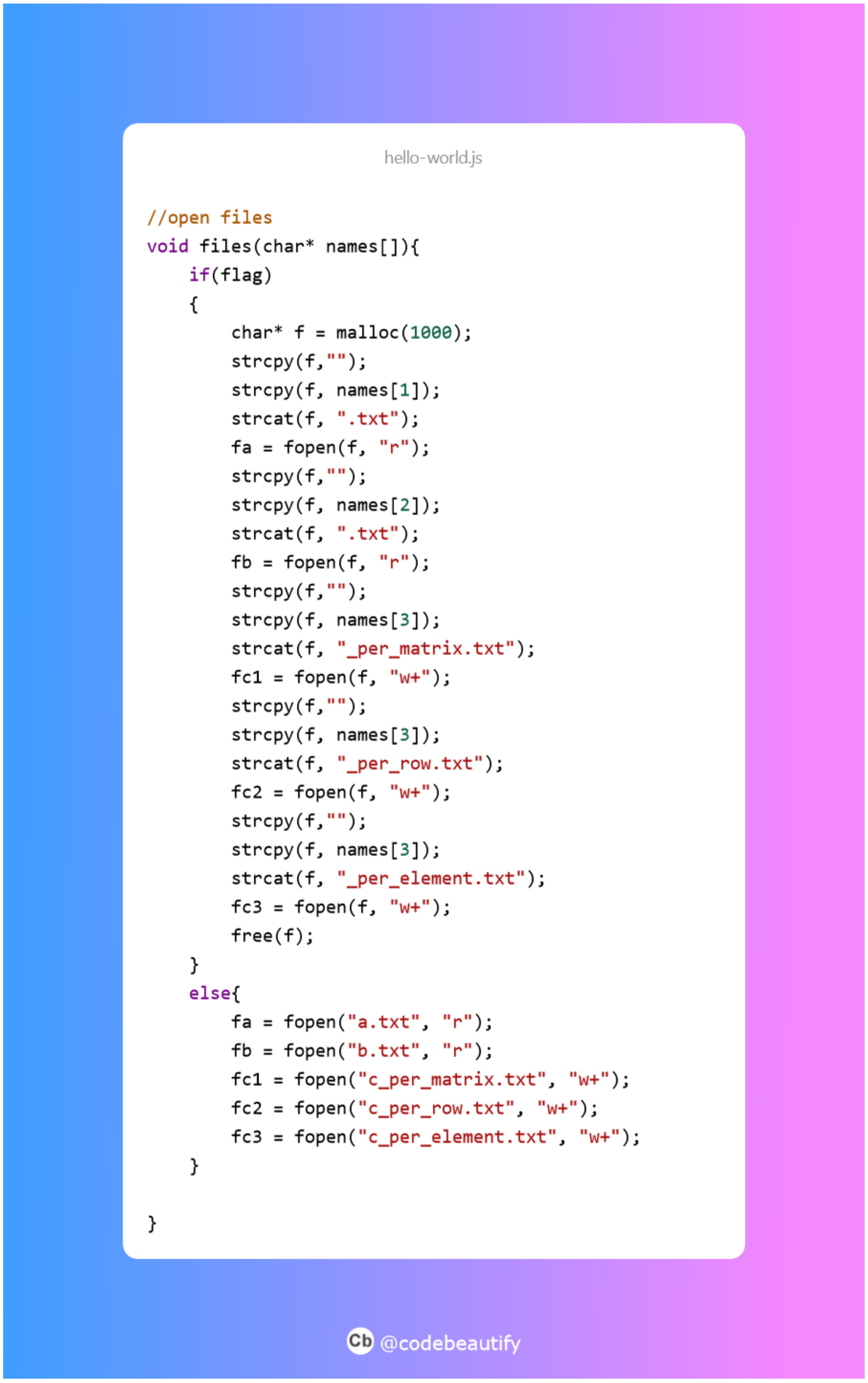
```
//take input from files
void read(int arr[100][100], FILE* f,int z){
    int row,colum;
    if(z)
    {fscanf(f, "row=%d col=%d", &row2, &colum2);
        row=row2;
        colum=colum2;}
    else
    {fscanf(f, "row=%d col=%d", &row1, &colum1);
        row=row1;
        colum=colum1;}
    for(int i=0; i<row; i++)
        for(int j=0; j<colum; j++)
            fscanf(f, "%d", &arr[i][j]);
}
//print output in files
void print(int arr[100][100], FILE* f){
    fprintf(f, "row=%d col=%d\n", row1, colum2);
    for(int i=0; i<row1; i++)
    {for(int j=0; j<colum2; j++)
            fprintf(f, "%d ", arr[i][j]);
    fprintf(f, "\n");}
    fclose(f);
}
```



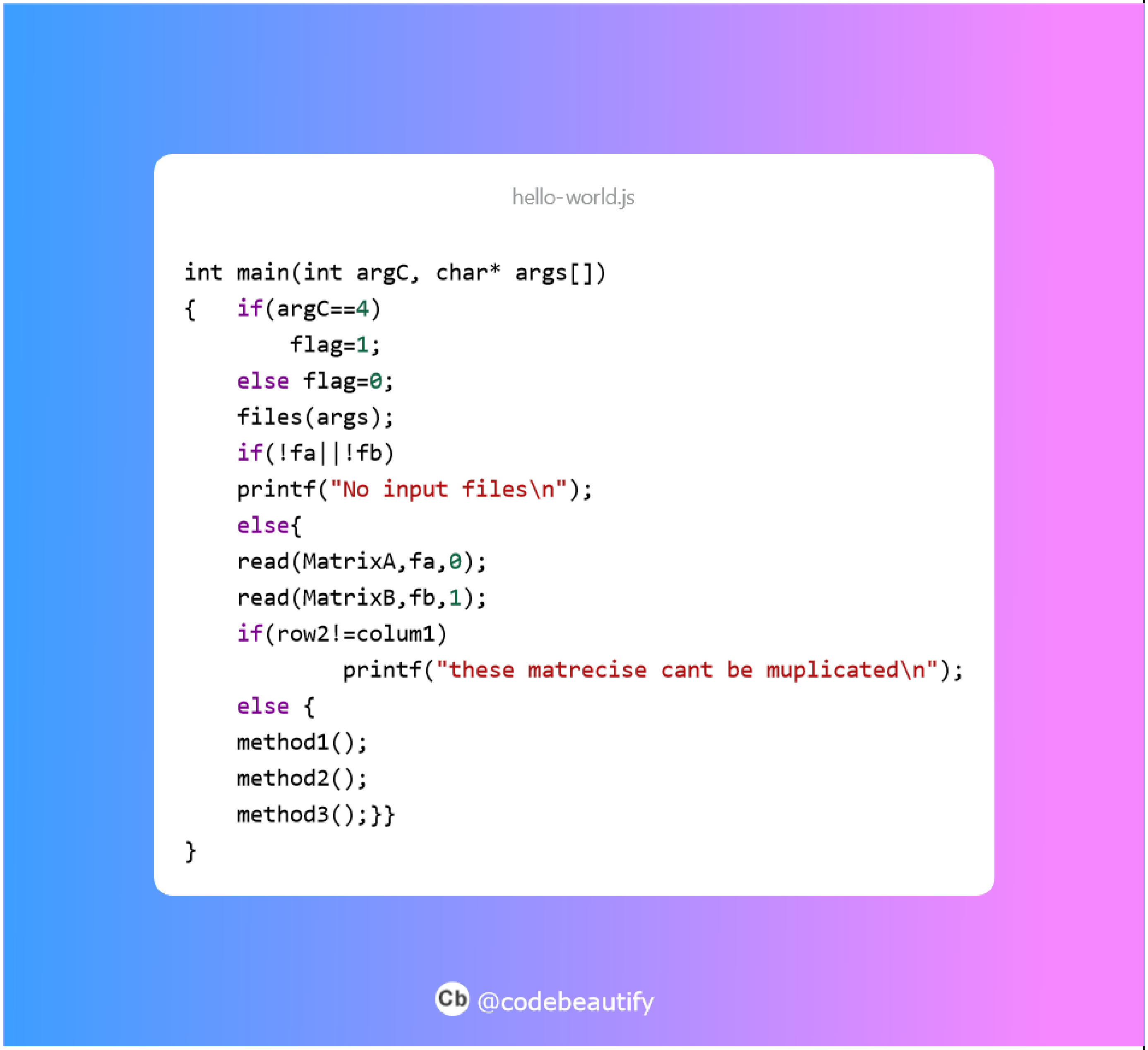
```
/calculate all elements
void *case1(){
    for(int i=0;i<row1;i++)
        for(int j=0;j<colum2;j++){
            MatrixC1[i][j]=0;
            for(int k=0;k<row2;k++)
                MatrixC1[i][j] += MatrixA[i][k]*MatrixB[k][j];

        }
}
//calculate one row
void *case2(void *r){
    long row=(long)r;
    for(int j=0;j<colum2;j++){
        MatrixC2[row][j]=0;
        for(int k=0;k<row2;k++)
            MatrixC2[row][j] += MatrixA[row][k]*MatrixB[k][j];

    }

}
//calculate one element
void *case3(void *u){
    struct rc *i;
    i = (struct rc *) u;
    int r=i->row, c=i->col;
    MatrixC3[r][c]=0;
    for(int k=0;k<row2;k++)
        MatrixC3[r][c] += MatrixA[r][k]*MatrixB[k][c];
    free(i);

}
```


hello-world.js

```
void method1(){
    printf("Method1:\n");
    struct timeval stop, start;
    gettimeofday(&start, NULL); //start checking time
    case1();
    gettimeofday(&stop, NULL); //end checking time
    printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
    printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);
    printf("Number of Threads Created: 1 Thread\n");
    fprintf(fc1, "Method: A thread per matrix\n");
    print(MatrixC1,fc1);
}
//making threads for case 2
void method2(){
    printf("Method2:\n");
    struct timeval stop, start;
    gettimeofday(&start, NULL); //start checking time
    pthread_t th[row1];

    for(int i=0;i<row1;i++)
        pthread_create(&th[i], NULL, case2, (void *)(long) i);

    for(int i=0;i<row1;i++)
        pthread_join(th[i], NULL);

    gettimeofday(&stop, NULL); //end checking time
    printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
    printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);
    printf("Number of Threads Created: %i Thread\n",row1);
    fprintf(fc2, "Method: A thread per row\n");
    print(MatrixC2,fc2);
}
//making threads for case 3
void method3(){
    printf("Method3:\n");
    struct timeval stop, start;
    gettimeofday(&start, NULL); //start checking time
    pthread_t th[row1][colum2];
    for(int i=0;i<row1;i++)
        for(int j=0;j<colum2;j++) {
            struct rc *a = malloc(sizeof(struct rc));
            a->row = i;
            a->col = j;
            pthread_create(&th[i][j], NULL, case3, (void *) a);
        }
    for(int i=0;i<row1;i++)
        for(int j=0;j<colum2;j++)
            pthread_join(th[i][j], NULL);

    gettimeofday(&stop, NULL); //end checking time
    printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
    printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);
    printf("Number of Threads Created: %i Thread\n",row1*colum2);
    fprintf(fc3, "Method: A thread per element\n");
    print(MatrixC3,fc3);
}
```


hello-world.js

```
//open files
void files(char* names[]){
    if(flag)
    {
        char* f = malloc(1000);
        strcpy(f,"");
        strcpy(f, names[1]);
        strcat(f, ".txt");
        fa = fopen(f, "r");
        strcpy(f,"");
        strcpy(f, names[2]);
        strcat(f, ".txt");
        fb = fopen(f, "r");
        strcpy(f,"");
        strcpy(f, names[3]);
        strcat(f, "_per_matrix.txt");
        fc1 = fopen(f, "w+");
        strcpy(f,"");
        strcpy(f, names[3]);
        strcat(f, "_per_row.txt");
        fc2 = fopen(f, "w+");
        strcpy(f,"");
        strcpy(f, names[3]);
        strcat(f, "_per_element.txt");
        fc3 = fopen(f, "w+");
        free(f);
    }
    else{
        fa = fopen("a.txt", "r");
        fb = fopen("b.txt", "r");
        fc1 = fopen("c_per_matrix.txt", "w+");
        fc2 = fopen("c_per_row.txt", "w+");
        fc3 = fopen("c_per_element.txt", "w+");
    }

}
```



```
int main(int argC, char* args[])
{   if(argC==4)
        flag=1;
    else flag=0;
    files(args);
    if(!fa||!fb)
    printf("No input files\n");
    else{
    read(MatrixA,fa,0);
    read(MatrixB,fb,1);
    if(row2!=colum1)
            printf("these matrecise cant be muplicated\n");
    else {
    method1();
    method2();
    method3();}}
}
```

How to run code:

In teminal type :

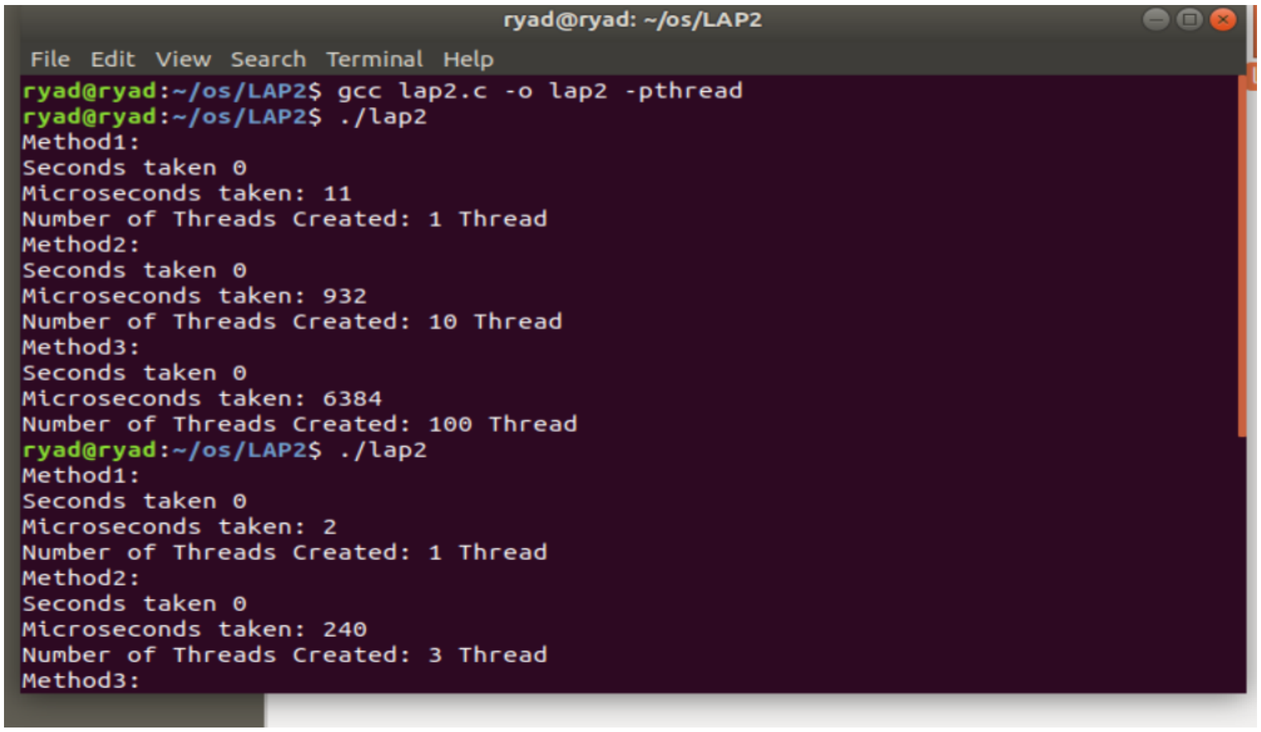
Gcc lap2.c -o lap2 -pthread

./lap2

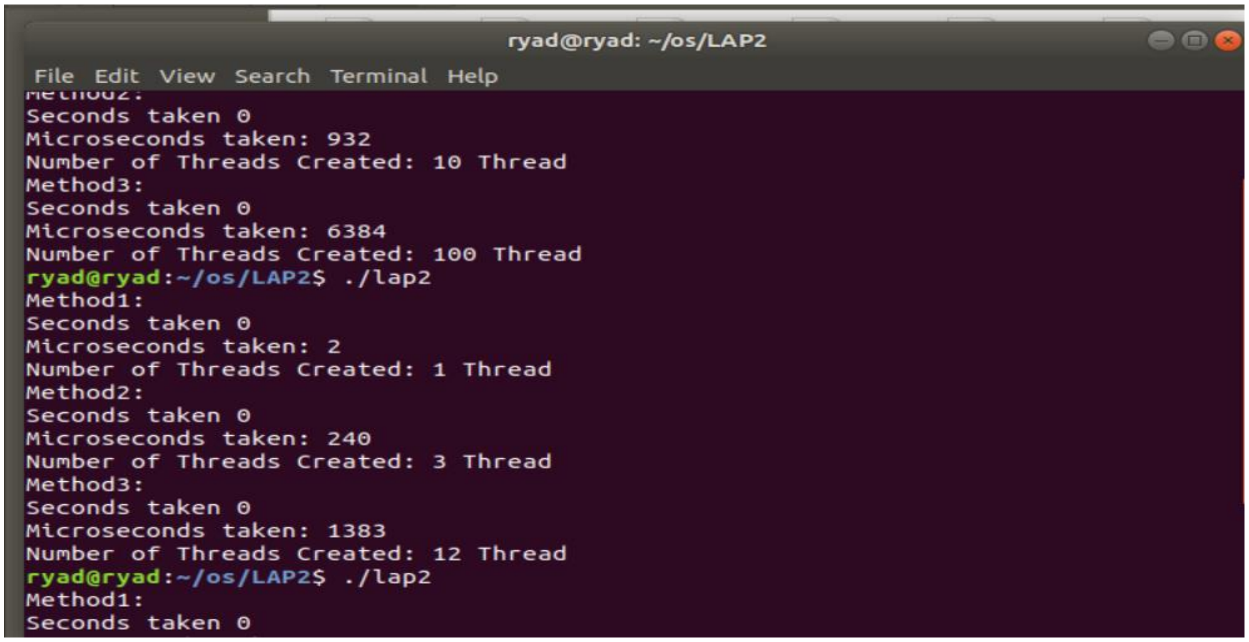
And you can give it arguments

./lap2 x y z

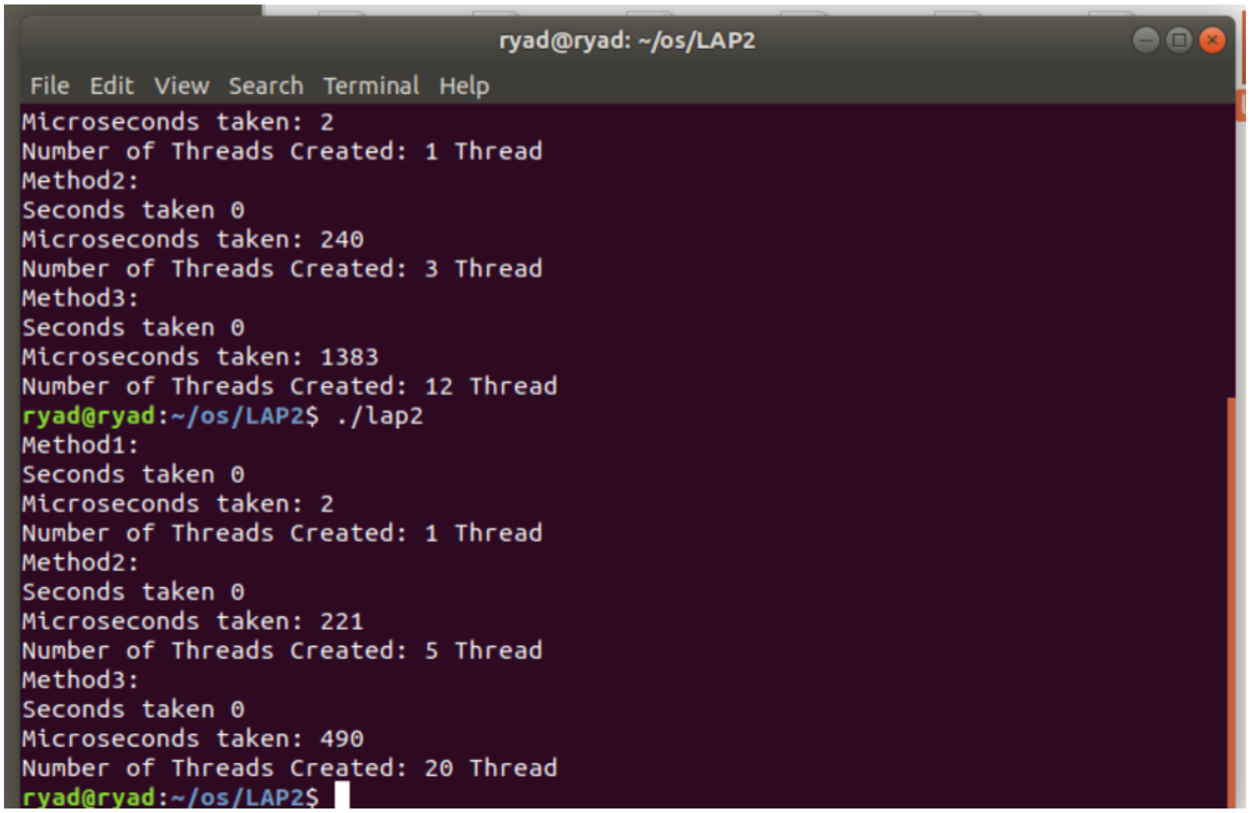
Sample runs:



```
ryad@ryad:~/os/LAP2$ gcc lap2.c -o lap2 -pthread
ryad@ryad:~/os/LAP2$ ./lap2
Method1:
Seconds taken 0
Microseconds taken: 11
Number of Threads Created: 1 Thread
Method2:
Seconds taken 0
Microseconds taken: 932
Number of Threads Created: 10 Thread
Method3:
Seconds taken 0
Microseconds taken: 6384
Number of Threads Created: 100 Thread
ryad@ryad:~/os/LAP2$ ./lap2
Method1:
Seconds taken 0
Microseconds taken: 2
Number of Threads Created: 1 Thread
Method2:
Seconds taken 0
Microseconds taken: 240
Number of Threads Created: 3 Thread
Method3:
```



```
Method2:
Seconds taken 0
Microseconds taken: 932
Number of Threads Created: 10 Thread
Method3:
Seconds taken 0
Microseconds taken: 6384
Number of Threads Created: 100 Thread
ryad@ryad:~/os/LAP2$ ./lap2
Method1:
Seconds taken 0
Microseconds taken: 2
Number of Threads Created: 1 Thread
Method2:
Seconds taken 0
Microseconds taken: 240
Number of Threads Created: 3 Thread
Method3:
Seconds taken 0
Microseconds taken: 1383
Number of Threads Created: 12 Thread
ryad@ryad:~/os/LAP2$ ./lap2
Method1:
Seconds taken 0
```



```
Microseconds taken: 2
Number of Threads Created: 1 Thread
Method2:
Seconds taken 0
Microseconds taken: 240
Number of Threads Created: 3 Thread
Method3:
Seconds taken 0
Microseconds taken: 1383
Number of Threads Created: 12 Thread
ryad@ryad:~/os/LAP2$ ./lap2
Method1:
Seconds taken 0
Microseconds taken: 2
Number of Threads Created: 1 Thread
Method2:
Seconds taken 0
Microseconds taken: 221
Number of Threads Created: 5 Thread
Method3:
Seconds taken 0
Microseconds taken: 490
Number of Threads Created: 20 Thread
ryad@ryad:~/os/LAP2$
```

A comparison between the three methods of matrix multiplication:
First method is faster than second method.
Second method is faster than third method.
The idea is, creating and handling threads requires extra overhead and lots of computation, so, it's not always the ideal solution to go for multi-threading, and there's always a tradeoff.
In designing multi-threaded programs, you always consider many things, including the threading overhead, the size of the problem, and the level of concurrency. There's no rule for that, you just make your own analysis and take your decision accordingly.
First method take threads less than second method.
second method take threads less than third method.