Operating System

Lab2

Name: Mahmoud Ebrahim Elsayed Mahmoud Manfy

Id: 57

Content:

- Basic idea.
- Main function.
- How to compile and run.
- Sample runs.
- Comparison between the two methods.

Basic idea:

- Take the inputs from the arguments of the main function.
- Open the input files and take the input with freopen function.
- Multiplies the two matrices with two methods using the threads:
    - First method: create a thread for each row:
        - Allocate storage for the answer matrix.
        - Loop on the rows.
        - build struct for each thread which carries the first matrix row, the whole second matrix, the result matrix row, the number of rows and columns for second matrix.
        - Path the struct for the function and compute the row.
        - Make the main thread wait all threads to finish.
        - Clean up the finished thread memory.
        - Print the result matrix in the output file.
    - Second method: creates a thread for each element:
        - Allocate storage for answering matrix.
        - Loop on each element on the result matrix
        - Build struct for each thread which carries the first matrix row, the whole second matrix, the element on the result matrix which wanted to compute, the index column for the second matrix and the number of rows for the second matrix.
        - Path the struct for the function and compute the element.
        - Make the main thread wait all threads to finish.
        - Clean up the finished thread memory.
        - Print the result matrix in the output file.

Main functions:

- **Method one:**

main - Notepad

File Edit Format View Help

```
// the function which passed to the thread
void * fun1(void * thread) {

    struct data1 *data = (struct data1 *)thread;
    int *a = data->a;
    int **b = data->b;
    int *c = data->c;
    int row = data->num_row;
    int col = data->num_col;

    for (int i = 0; i < col; ++i) {
        c[i] = 0;
        for (int j = 0; j < row; ++j) {
            c[i] += a[j]*b[j][i];
        }
    }

    pthread_exit(0);
}
```

The function which corresponds the threads and takes data1 as input.

main - Notepad

File Edit Format View Help

```
// stores the data used in function
struct data1{
    int **b; // the whole mat2
    int *a; // the row in mat1
    int *c; // the row in result mat
    int num_row; // the number of row
    int num_col; // the number of col
};
```

The struct holds the data for the threads.

main - Notepad

File Edit Format View Help

```
    pthread_t threads[num_thread]; struct data1 *data1[num_thread]; // to store the struct which still in memory to clean up it

    for (int i = 0; i < num_thread; ++i) {

        struct data1 *data = malloc(sizeof(struct data1));
        data->a = mat1[i];
        data->b = mat2;
        data->c = res[i];
        data->num_row = y;
        data->num_col = z;
        int error = pthread_create(&threads[i],NULL,fun1,(void *) data);
        if (error) {
            printf("ERROR; return code from pthread_create() is %d\n", error);
            exit(-1);
        }
        data1[i] = data;
    }
```

Creates an array of threads, iterates over them and stores the data of the threads in the array to clean up it when finished.

```
        data1[i] = data;
    }

    for (int i = 0; i < num_thread; ++i) {
        pthread_join(threads[i],NULL); // wait for all threads to finish
        free(data1[i]); // clean space form finished threads
    }

    gettimeofday(&stop, NULL); // end time
```

This loop makes the parent thread wait all the threads to finish and clean up the storage of finished thread.

- **Method two:**

```
// function used in the thread
void * fun2(void * thread) {

    struct data2 *data = (struct data2 *)thread;
    int *a = data->a;
    int **b = data->b;
    int *c = data->c;
    int row = data->num_row;
    int index = data->num_col;
    *c = 0; // make the element with zero in the marix
    for (int i = 0; i < row; ++i) {
        *c += a[i]*b[i][index];
    }

    pthread_exit(0);
}
```

The function which corresponds the threads and takes data2 as input.

```
/*
 * work on method two
 */

struct data2{
    int **b; // the whole matrix two
    int *a; // the row in matrix one
    int *c; // element in result matrix
    int num_row; // the number of row
    int num_col; // the column number for matrix two
};
```

The struct holds the data for the threads.

main - Notepad

File Edit Format View Help

```
    }

    pthread_t threads[x][z]; struct data2 *data2[x][z];

    gettimeofday(&start, NULL);

    for (int i = 0; i < x; ++i) {
        for (int j = 0; j < z; ++j) {
            struct data2 *data = malloc(sizeof(struct data2));

            data->a = mat1[i];
            data->b = mat2;
            data->c = &res[i][j];
            data->num_row = y;
            data->num_col = j;

            int error = pthread_create(&threads[i][j],NULL,fun2,(void *) data);
            if (error) {
                printf("ERROR; return code from pthread_create() is %d\n", error);
                exit(-1);
            }

            data2[i][j] = data;
        }
    }
```

Creates an array of threads, iterates over them and stores the data of the threads in the array to clean up it when finished.

main - Notepad

File Edit Format View Help

```
    }

    for (int i = 0; i < x; ++i) {
        for (int j = 0; j < z; ++j) {
            pthread_join(threads[i][j],NULL);
            free(data2[i][j]);
        }
    }

    gettimeofday(&stop, NULL);
```

This loop makes the parent thread wait all the threads to finish and clean up the storage of finished thread.

How to compile and run:

- Put the main file “main.c” in the same folder with Makefile.
- Open the terminal and go to the directory of Makefile.
- write the command “make”.
- write the command “./matmult firstFileName secondFileName outputFileName”
- If you didn’t enter any file name, the default names will be taken.
- Now your program is running.

Sample Runs:

The first file content:

a.txt
~/CLionProjects/Lab2

```
row=5 col=5
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
```

The second file content:

b.txt
~/CLionProjects/Lab2

a.txt b.txt

```
row=5 col=4
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16
17      18      19      20
```

The output file content:

c.out
~/CLionProjects/Lab2

a.txt | b.txt | c.out

```
Method1 result

175		190		205		220
400		440		480		520
625		690		755		820
850		940		1030		1120
1075		1190		1305		1420


Method2 result

175		190		205		220
400		440		480		520
625		690		755		820
850		940		1030		1120
1075		1190		1305		1420
```

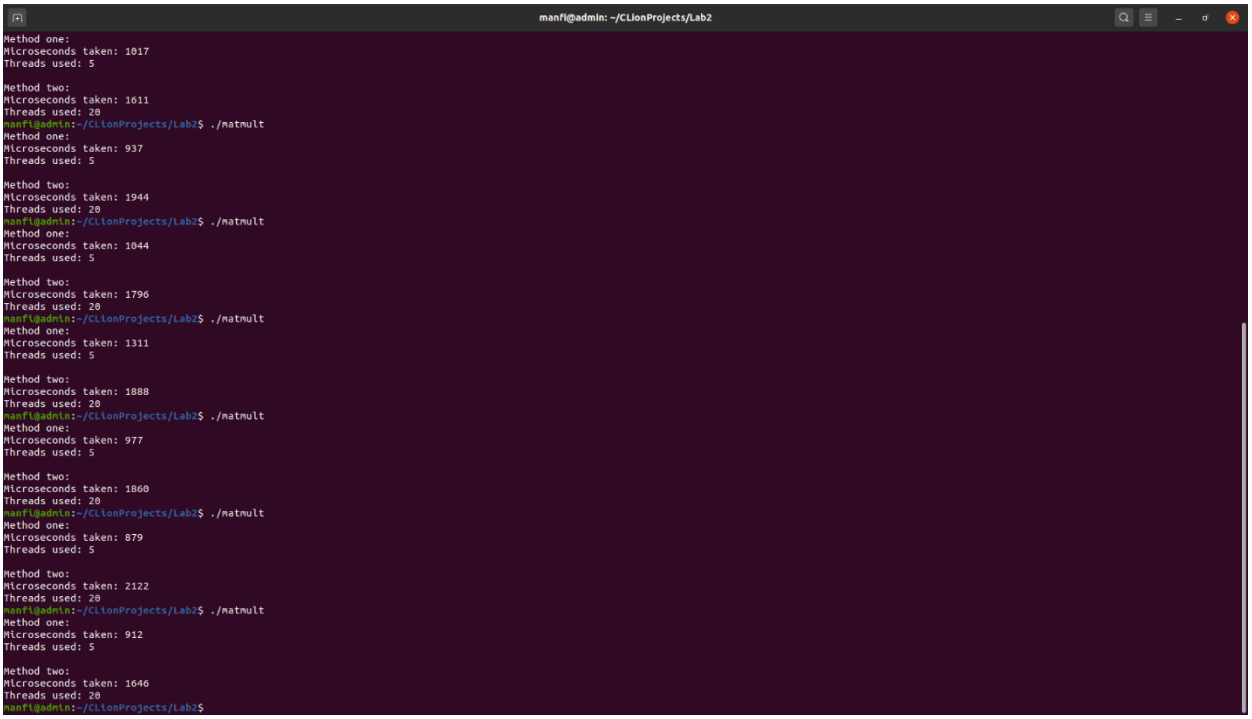
The terminal output is:

```
Threads used: 5

Method two:
Microseconds taken: 1687
Threads used: 20
manfi@admin:~/CLionProjects/Lab2$ ./matmult a.txt b.txt c.out
Method one:
Microseconds taken: 267
Threads used: 5

Method two:
Microseconds taken: 428
Threads used: 20
```

Comparison between the two methods:



From the output, we noticed that the first method is faster than the second method, because the number of threads in the second method is larger.