

Operating System

Lab Two Threads

```
/ # root
├── /usr # shareable and read-only # data
│   ├── /local # local software
│   │   ├── /bin
│   │   ├── /games
│   │   ├── /include
│   │   ├── /lib
│   │   ├── /man
│   │   ├── /sbin
│   │   ├── /share
│   │   ├── /src
│   │   ├── /share
│   │   ├── /static data sharable # among all architectures
│   │   ├── /man # manual pages
│   │   ├── /man1 # user programs
│   │   ├── /man2 # system calls
│   │   ├── /man3 # lib functions
│   │   ├── /man4 # special file
│   │   ├── /man5 # file formats
│   │   ├── /man6 # games
│   │   ├── /man7 # misc.
│   │   ├── /man8 # system admin.
│   │   ├── /bin # most user commands
│   │   ├── /include
│   │   ├── /lib # standard include files # for 'C' prog.
│   │   ├── /lib # obj. bin, lib files # for prog. and packages
│   │   ├── /sbin # non essential binaries
│   ├── /cache # application cache data
│   ├── /lib # variable state data # remains after reboot
│   ├── /yp # data for yps services
│   ├── /lock # lock files for shared # resources
│   ├── /opt # variable data of # packages installed
│   ├── /run # info of system since it # was booted
│   ├── /tmp # available for prog.
│   ├── /spool # data awaiting processing
│   ├── /lpd
│   ├── /mqueue
│   ├── /news
│   ├── /rwho
│   ├── /uucp
│   ├── /log # log files and dir
│   ├── /lastlog
│   ├── /messages
│   └── /wtmp
├── /var # variable data files
├── /sbin # system binaries
│   ├── fastboot
│   ├── fastboot
│   ├── fastboot
│   ├── halt
│   ├── ifconfig
│   ├── init
│   ├── mkfs
│   ├── mkswap
│   ├── reboot
│   ├── route
│   ├── swapoff
│   └── update
├── /tmp # temporary files deleted on # reboot
├── /dev # location of special or # device files # [contains makedev]
├── /home # user home directories
├── /lib # library and kernel modules
├── /mnt # mount files for temporary # filesystems
├── /opt # add-on application # filesystems
└── /root # home dir. for root user
```

Presented by:

Mostafa Hakam

5525

CONTENTS

Summary2

Code3

Sample Runs8

Implementing two popular algorithms as multi-threaded ones:

1. Matrix Multiplication
2. Merge Sort

Language Used:

- C/C++ Language.

Libraries used:

1. Library "unistd".
2. Library "stdio".
3. Library "pthread".
4. Library "sys/time".
5. Library "stdlib".

I. Matrix Multiplication:

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/time.h>

typedef struct {
    int ** mat_one;
    int ** mat_two;
    int ** mat_output;
    int r2;
}threadElement;

typedef struct {
    int i, j;
}indexes;

typedef struct {
    threadElement* myThreadElement;
    indexes* myIndexes;
}threadArgs;

void* elementMultiplication(void* args) {

    threadArgs * myThreadArgs = (threadArgs *) args;
    threadElement* myThread = myThreadArgs->myThreadElement;

    int i = myThreadArgs->myIndexes->i;
    int j = myThreadArgs->myIndexes->j;

    myThread->mat_output[i][j] = 0;

    int k;
    for(k = 0; k < myThread->r2 ; k++)
        myThread->mat_output[i][j] += (myThread->mat_one[i][k] * myThread->mat_two[k][j]);

    pthread_exit(NULL);
}

void* rowMultiplication(void* args) {
    threadArgs * myThreadArgs = (threadArgs *) args;
    threadElement* myThread = myThreadArgs->myThreadElement;

    int i = myThreadArgs->myIndexes->i;

    int k, l;
    for(k = 0; k < myThread->r2; k++) {
        myThread->mat_output[i][k] = 0;
        for (l = 0; l < myThread->r2; ++l) {
            myThread->mat_output[i][k] += (myThread->mat_one[i][l] * myThread->mat_two[l][k]);
        }
    }

    pthread_exit(NULL);
}

int main() {
    int r1, c1, r2, c2, i, j;
    struct timeval start, stop;

    // Opening File Containing inputs
    FILE *fptr = fopen("input.txt", "r");

    // Reading First Matrix
    fscanf(fptr, "%d %d", &r1, &c1);

    int *matrix_one[r1];
    for (i = 0; i < r1; i++)
        matrix_one[i] = (int *)malloc(c1 * sizeof(int));
```

```

    for (i = 0; i < r1; i++)
        for (j = 0; j < c1; j++)
            fscanf(fptr, "%d", &matrix_one[i][j]);

    // Reading Second Matrix
    fscanf(fptr, "%d %d", &r2, &c2);

    int *matrix_two[r2];
    for (i = 0; i < r2; i++)
        matrix_two[i] = (int *)malloc(c2 * sizeof(int));

    for (i = 0; i < r2; i++)
        for (j = 0; j < c2; j++)
            fscanf(fptr, "%d", &matrix_two[i][j]);
    fclose(fptr);

    // Printing First Matrix
    printf("First Matrix:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c1; j++)
            printf("%d ", matrix_one[i][j]);
        printf("\n");
    }

    // Printing Second Matrix
    printf("\nSecond Matrix:\n");
    for (i = 0; i < r2; i++) {
        for (j = 0; j < c2; j++)
            printf("%d ", matrix_two[i][j]);
        printf("\n");
    }

    int *matrix_output[r1];
    for (i = 0; i < r1; i++)
        matrix_output[i] = (int *)malloc(c2 * sizeof(int));

    pthread_t threads[r1][c2];

    threadElement * myThreadElement = malloc(sizeof(threadElement));

    myThreadElement->mat_one = matrix_one;
    myThreadElement->mat_two = matrix_two;
    myThreadElement->mat_output = matrix_output;
    myThreadElement->r2 = r2;

    int error;
    indexes * indexesStruct = (indexes *)malloc(r1 * c1 * sizeof(indexes));
    threadArgs * threadArgsStruct = (threadArgs *)malloc(r1 * c1 * sizeof(threadArgs));

    // Element by Element Algorithm
    // Start checking time
    gettimeofday(&start, NULL);

    for(i = 0; i < r1; i++) {
        for(j = 0; j < c2; j++) {
            indexesStruct[i * c2 + j].i = i;
            indexesStruct[i * c2 + j].j = j;

            threadArgsStruct[i * c2 + j].myThreadElement = myThreadElement;
            threadArgsStruct[i * c2 + j].myIndexes = &indexesStruct[i * c2 + j];

            threads[i][j] = malloc(sizeof(pthread_t));

            error = pthread_create(&threads[i][j], NULL, elementMultiplication, (void *) &threadArgsStruct[i * c2 + j]);

            if(error)
                printf("\nError creating thread... \n");
        }
    }

    for(i = 0; i < r1; i++) {
        for(j = 0; j < c2; j++) {
            pthread_join(threads[i][j], NULL);

```

```

    }
}
gettimeofday(&stop, NULL);

fptr = fopen("output.txt", "w+");

// Printing Output Matrix
printf("\nOutput Matrix (Element by Element):\n");
fprintf(fptr, "Matrix Output 1:\n");

for (i = 0; i < r1; i++) {
    for (j = 0; j < c2; j++) {
        printf("%d ", matrix_output[i][j]);
        fprintf(fptr, "%d ", matrix_output[i][j]);
    }
    printf("\n");
    fprintf(fptr, "\n");
}

printf("\nTime (in micro-seconds) taken for element by element: %lu\nNumber of threads created = %d\n",
stop.tv_usec - start.tv_usec, r1 * c2);
fprintf(fptr, "END1\t[%d]\n\n", stop.tv_usec - start.tv_usec);

// Row by Row Algorithm
// Start checking time
gettimeofday(&start, NULL);

for(i = 0; i < r1; i++) {
    indexesStruct[i].i = i;

    threadArgsStruct[i].myThreadElement = myThreadElement;
    threadArgsStruct[i].myIndexes = &indexesStruct[i];

    error = pthread_create(&threads[i][0], NULL, rowMultiplication, (void *) &threadArgsStruct[i]);
    if(error)
        printf("\nError creating thread... \n");
}

for(i = 0; i < r1; i++) {
    pthread_join(threads[i][0], NULL);
}

gettimeofday(&stop, NULL);

// Printing Output Matrix
printf("\nOutput Matrix (Row by Row):\n");
fprintf(fptr, "Matrix Output 2:\n");

for (i = 0; i < r1; i++) {
    for (j = 0; j < c2; j++) {
        printf("%d ", matrix_output[i][j]);
        fprintf(fptr, "%d ", matrix_output[i][j]);
    }
    printf("\n");
    fprintf(fptr, "\n");
}

printf("\nTime (in micro-seconds) taken for row by row: %lu\nNumber of threads created = %d\n", stop.tv_usec -
start.tv_usec, r1);
fprintf(fptr, "END2\t[%d]", stop.tv_usec - start.tv_usec);

fclose(fptr);

return 0;
}

```

II. Merge Sort:

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

typedef struct {
    int* arr;
    int l;
    int r;
} mergeArgs;

void merge(int * arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int* arr, int l, int r) {
    if (l < r) {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

void* mergeSortThread(void * args) {
    mergeArgs* myArgs = (mergeArgs *) args;

    int * arr = &myArgs->arr[0];

    int l = myArgs->l;
```

```

int r = myArgs->r;

if (l < r) {
    pthread_t * firstThread = malloc(sizeof(pthread_t));
    pthread_t * secondThread = malloc(sizeof(pthread_t));
    mergeArgs * myArgsOne = malloc(sizeof(mergeArgs));
    mergeArgs * myArgsTwo = malloc(sizeof(mergeArgs));

    int m = l + (r - l) / 2;

    myArgsOne->arr = &arr[0];
    myArgsOne->l = l;
    myArgsOne->r = m;

    pthread_create(&firstThread, NULL, mergeSortThread, myArgsOne);

    myArgsTwo->arr = &arr[0];
    myArgsTwo->l = m + 1;
    myArgsTwo->r = r;

    pthread_create(&secondThread, NULL, mergeSortThread, myArgsTwo);

    pthread_join(firstThread, NULL);
    pthread_join(secondThread, NULL);

    merge(arr, l, m, r);
}
pthread_exit(NULL);
}

void main() {
    FILE *fptr = fopen("input.txt", "r");

    int i, size;
    mergeArgs myArgs;

    // Reading Array
    fscanf(fptr, "%d", &size);

    int * array = malloc(size * sizeof(int));
    int * arrayThread = malloc(size * sizeof(int));

    for(i = 0; i < size; i++) {
        fscanf(fptr, "%d", &array[i]);
        arrayThread[i] = array[i];
    }

    // Printing Unsorted Array
    printf("Unsorted Array:\n");
    for(i = 0; i < size; i++)
        printf("%d ", array[i]);

    mergeSort(array, 0, size - 1);

    // Printing Sorted Array
    printf("\n\nSorted Array (no threads):\n");
    for(i = 0; i < size; i++)
        printf("%d ", array[i]);
    fclose(fptr);

    pthread_t * myThread = malloc(sizeof(pthread_t));

    myArgs.arr = arrayThread;
    myArgs.l = 0;
    myArgs.r = size - 1;

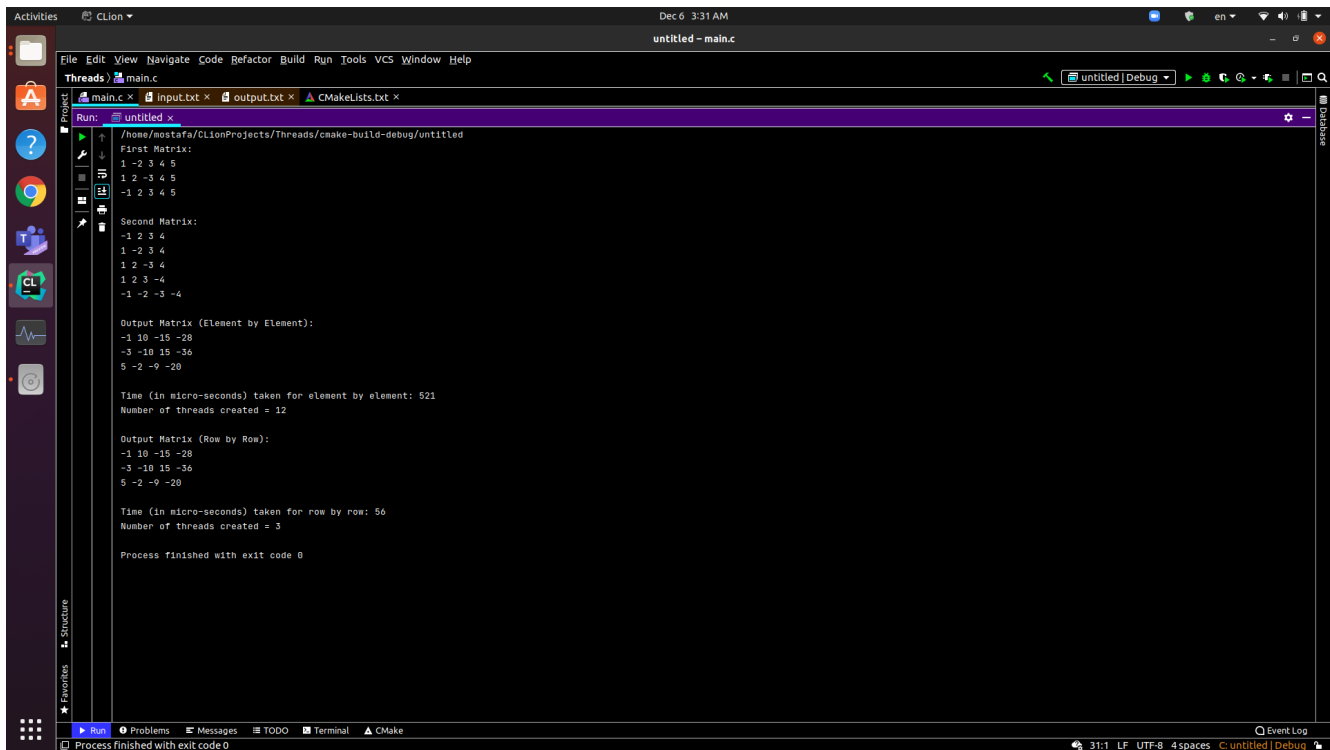
    pthread_create(&myThread, NULL, mergeSortThread, (void *) &myArgs);
    pthread_join(myThread, NULL);

    // Printing Sorted Array
    printf("\n\nSorted Array (with threads):\n");
    for(i = 0; i < size; i++)
        printf("%d ", arrayThread[i]);
}

```


Sample Runs

Matrix Multiplication:



```

/home/mostafa/CLionProjects/Threads/cmake-build-debug/untitled
First Matrix:
1 -2 3 4 5
1 2 -3 4 5
-1 2 3 4 5

Second Matrix:
-1 2 3 4
1 -2 3 4
1 2 -3 4
1 2 3 -4
-1 -2 -3 -4

Output Matrix (Element by Element):
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20

Time (in micro-seconds) taken for element by element: 521
Number of threads created = 12

Output Matrix (Row by Row):
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20

Time (in micro-seconds) taken for row by row: 56
Number of threads created = 3

Process finished with exit code 0

```

Input file (input.txt)

```

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

```

Output file (output.txt)

Matrix Output 1:

```

-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
END1      [521]

```

Matrix Output 2:

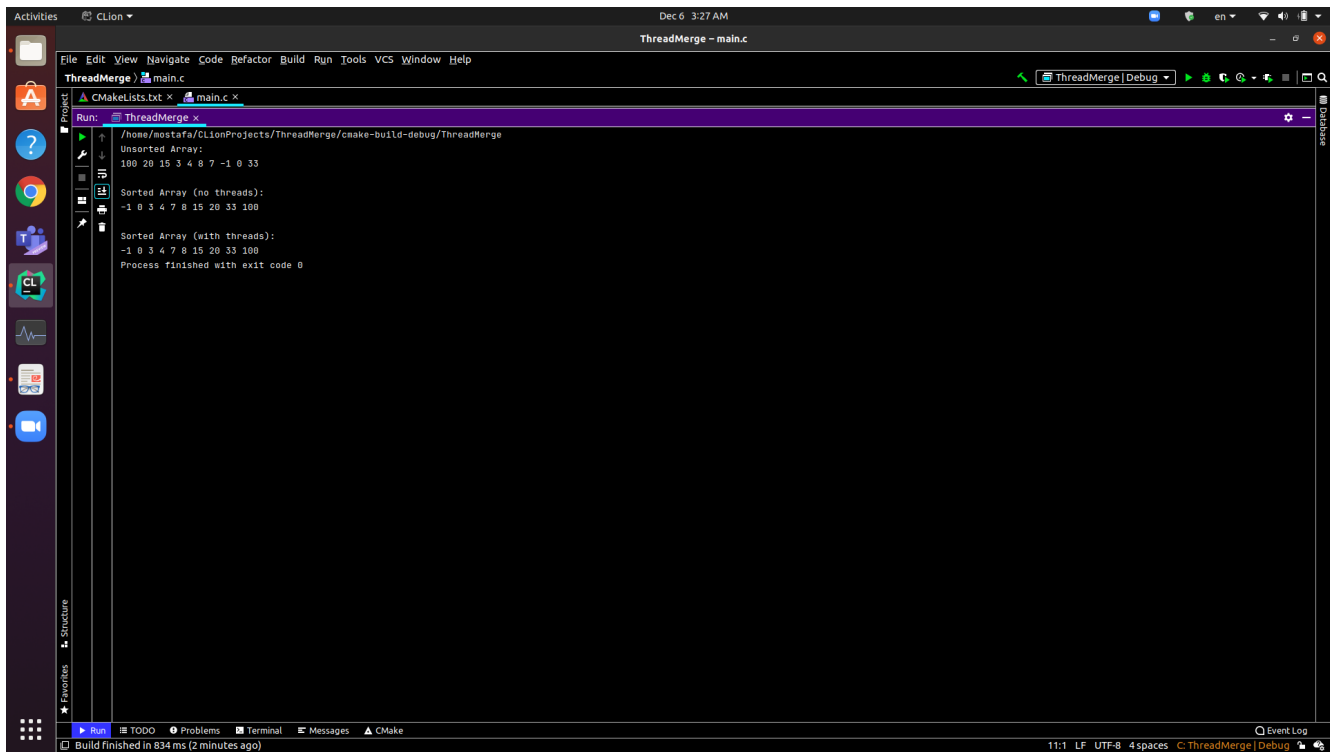
```

-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
END2      [56]

```

Sample Runs

Merge Sort:



```
ThreadMerge - main.c
Run: ThreadMerge x
/home/mostafa/CLionProjects/ThreadMerge/cmake-build-debug/ThreadMerge
Unsorted Array:
100 20 15 3 4 8 7 -1 0 33
Sorted Array (no threads):
-1 0 3 4 7 8 15 20 33 100
Sorted Array (with threads):
-1 0 3 4 7 8 15 20 33 100
Process finished with exit code 0
```