System Programming

Homework 5 Report | 2021 - 2022

Ahmet Tuğkan Ayhan

1901042692

Command Line Arguments

A Valid Example : “ ./hw5 -i "input1.txt" -j "input2.txt" -o "output.csv" -n 3 -m 4 ”

Input & Output Files

There are 2 input files and 1 output file. If output file not exist while executing the code, then it will be created during saving operations.

N & M Values

n value is power value for 2 and matrix size that will be read from the input files is $2^n x 2^n$.
n value must be bigger than 2

```
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
gcc -ggdb3 -Wall source/main.c source/file.c -o hw5 -lpthread -lm &&
 ./hw5 -i "input1.txt" -j "input2.txt" -o "output.csv" -n 1 -m 4
n value must be bigger than 2.
Success
make: *** [Makefile:10: run] Error 1
tgknyhn@tgknyhn:~/Desktop/hw5$
```

m value shows how many threads will be created. And in order to equally divide threads, **m** value must be equal to or bigger than 2. Most importantly, **m** value must be a power of 2

```
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
gcc -ggdb3 -Wall source/main.c source/file.c -o hw5 -lpthread -lm &&
 ./hw5 -i "input1.txt" -j "input2.txt" -o "output.csv" -n 3 -m 1
M must be a power of 2 and must be bigger than 2.
Success
make: *** [Makefile:10: run] Error 1
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
gcc -ggdb3 -Wall source/main.c source/file.c -o hw5 -lpthread -lm &&
 ./hw5 -i "input1.txt" -j "input2.txt" -o "output.csv" -n 3 -m 5
M must be a power of 2 and must be bigger than 2.
Success
make: *** [Makefile:10: run] Error 1
tgknyhn@tgknyhn:~/Desktop/hw5$
```

After cla controlling done, I assigned these values to global variables for easy use.

Signal Handling

Even before controlling command line arguments, I initialized signal handler. After that, I checked every function looking for sigint signal before executing it.

```
int main(int argc, char *argv[]) {
    // First, initialize signal handler for SIGINT signal
    initializeSignalHandler();
    // Start Timer
    if(sigintFlag == NOSIGNAL) gettimeofday(&start, NULL);
    // Control command line arguments
    if(sigintFlag == NOSIGNAL) ctrlcla(argc, argv);
```

initializeSignalHandler sets **handleSIG** function as signal handler function and inside **handleSIG** it only looks for signal type and if it is **SIGINT** signal, then assigns **sigintFlag** from **NOSIGNAL** to **SIGNAL**. If I put a raise(SIGINT) after any function I will be warmed on console and given output file will be deleted, allocated matrices, other variables are deallocated and finally, program is terminated using exit(EXIT_FAILURE).

```
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
gcc -ggdb3 -Wall source/main.c source/file.c -o hw5 -lpthread -lm && ./hw5 -i "input1.txt" -j "i
nput2.txt" -o "output.csv" -n 3 -m 4
18/5/2022 | 15:15:33 -> Two matrices of size 8x8 have been read. The number of threads is 4
SIGINT signal detected. Terminating program after cleaning.
make: *** [Makefile:10: run] Error 1
tgknyhn@tgknyhn:~/Desktop/hw5$
```

Matrices

I used total of 4 2D matrix array and one buffer array for matrices. These matrices are:

1. **char* buffer :** Instead of reading one character and inserting it's ASCII value to **matrixA** and **matrixB** directly, I loaded every character to a **buffer** array. It's dimensions not known until runtime(size: **2^nx2^n**) because of that I used **malloc** to initialize it. For the **read** operation I used library that I created **file.c** and **file.h** . Inside these libraries read operation is done with basic syscalls(**open**, **read** and **close** functions).
2. **int** matrixA :** After buffer char array read first input file, it's contents are transferred to the **matrixA** after casting it using (int). Dimensions of **matrixA** is same with **matrixA[2^n][2^n]**

3. int** matrixB : After **matrixA** is done, buffer is filled with second file input again and same procedure is applied to this matrix also.
4. int** matrixC : **matrixC** value is calculated using **getMatrixC(index_t limit)** function. Here **getMatrixC** function is called inside threads. Because of that I created **index_t struct** giving it's **start column index** and **end column index.** By doing so with one parameter I was able to give 2 parameters which is required inside **getMatrixC** function. After function terminates **matrixC** is calculated.
5. double complex** matrixDFT : This matrix was the most complicated matrix to calculate. It's dimensions are as same as others($2^n x 2^n$) but the difficult part is the equation itself to get 2D Discrete Fourier Transform. We compared our results with my friends and get same results, so we are thinking that end result is correct. I created a function **calculate2DFourierTransform(double complex* dft, index_t limit).** Since I was not able to return 2D array from a thread, I converted 2D array into 1D and gave it as parameter(this is a function called inside a thread so in order to return dft array from thread, it must be given to this function as parameter). Inside this function I used this formula:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\iota 2\pi \left(\frac{ki}{N} + \frac{lj}{N}\right)}$$

And implemented it like that using **complex.h** library:

```
void calculate2DFourierTransform(double complex* dft, index_t limit) {
    int index = 0;
    for(int k=0; k<powN; k++) {
        for(int l=limit.start; l<=limit.end; l++, index++) {
            double realPart = 0;
            double imagPart = 0;
            for(int m=0; m<powN; m++) {
                for(int n=0; n<powN; n++) {
                    // Power value for e
                    double part1 = (-2) * (M_PI) * k * m /(double)powN;
                    double part2 = (-2) * (M_PI) * l * n /(double)powN;
                    // f(m,n)
                    double complex f_mn = matrixC[m][n];

                    realPart += f_mn * cos(part1 + part2);
                    imagPart += f_mn * 1.0 * sin(part1 + part2);
                }
            }
            dft[index] = realPart + imagPart * I;
        }
    }
}
```

Condition Variables and Mutexes

This part is very easy compared to calculating 2D Discrete Fourier Transform.

Instead of initializing condition variable and mutex dynamically, I used initializers to initialize statically.

```
pthread_mutex_t waitLock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t waitCond = PTHREAD_COND_INITIALIZER;
int finishedCount = 0;
```

Since problem is already explicitly written in the instructions pdf, I found same problem in the slides, which is synchronisation barrier, and implemented it. But with the difference, instead of using if, I used while loop.

```
void waitOthersOrBroadcast(int threadID) {
    // Inform the user
    gettimeofday(&end, NULL);
    printToConsole2(threadID, getElapsedTime(start, end));
    // Lock the mutex to enter critical region
    if(pthread_mutex_lock(&waitLock) != 0)
        exitWithError("Error occured while executing pthread_mutex_lock.\n");
    // Increment the finished thread count
    finishedCount++;     // Wait until everybody finishes
    while(finishedCount < M)
        pthread_cond_wait(&waitCond, &waitLock);
    // broadcast since this is the last thread
    pthread_cond_broadcast(&waitCond);
    // Inform the user
    printToConsole3(threadID);
    // Release the lock since we finished our job in critical region
    pthread_mutex_unlock(&waitLock);
}
```

This function is also called inside main thread function "**void* threadTasks(void* arg)**".

I stored **threads' id** value inside **index_t struct** together with column indexes.

What a Thread Does?

Since it is self-explanotary I will give a screenshot of my threadTasks function.

```
void* threadTasks(void* arg) {
    // First cast arg to index_t
    index_t limit = *((index_t*)arg);
    // Then calculate matrixC
    getMatrixC(limit);
    // Wait other threads or broadcast if it is the last one
    waitOthersOrBroadcast(limit.id);
    // Start the next timer
    gettimeofday(&startB, NULL);
    // Allocate space for 2D Fourier Transform applied matrix
    int row = powN, column = powN / M;
    double complex* dft = allocateDftMatrix(row, column);
    // Calculate the 2D Fourier Transform
    calculate2DFourierTransform(dft, limit);
    // Finish second timer and inform the user
    gettimeofday(&end, NULL);
    printToConsole4(limit.id, getElapsedTime(startB, end));
    // Return the 2D discrete fourier transform
    return (void*)dft;
}
```

This function simply called from main process with pthread_create and given argument is an index_t struct object. It first calculates matrixC with it's column indexes, then it waits until other threads finish their job.

When they finish, threads allocate space for their 1D dft matrix to return it in the end. These matrices are free'd at the end of main function so it is ok to allocate them here. After matrices are allocated, they calculate their part of 2D Fourier Transform and return it at the end.

Examples

I am using 8 core M1 chip on my computer and I will test my program with 3 different n and m values.

n = 3 , m = 2 , dimensions = 8 x 8

```
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
gcc -ggdb3 -Wall source/main.c source/file.c -o hw5 -lpthread -lm && ./hw5 -i "input1.txt" -j "input2.txt" -o "output.csv" -n 3 -m 2
18/5/2022 | 16:19:54 -> Two matrices of size 8x8 have been read. The number of threads is 2
18/5/2022 | 16:19:54 -> Thread 1 has reached the rendezvous point in 0.000102 seconds.
18/5/2022 | 16:19:54 -> Thread 2 has reached the rendezvous point in 0.000134 seconds.
18/5/2022 | 16:19:54 -> Thread 2 is advancing to the second part.
18/5/2022 | 16:19:54 -> Thread 1 is advancing to the second part.
18/5/2022 | 16:19:54 -> Thread 2 has finished the second part in 0.000033 seconds.
18/5/2022 | 16:19:54 -> Thread 1 has finished the second part in 0.000046 seconds.
18/5/2022 | 16:19:54 -> The process has written to the output file. The total time spent is 0.000515 seconds
tgknyhn@tgknyhn:~/Desktop/hw5$
```

n = 3 , m = 4 , dimensions = 8 x 8

```
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
gcc -ggdb3 -Wall source/main.c source/file.c -o hw5 -lpthread -lm && ./hw5 -i "input1.txt" -j "input2.txt" -o "output.csv" -n 3 -m 4
18/5/2022 | 16:22:41 -> Two matrices of size 8x8 have been read. The number of threads is 4
18/5/2022 | 16:22:41 -> Thread 2 has reached the rendezvous point in 0.000108 seconds.
18/5/2022 | 16:22:41 -> Thread 1 has reached the rendezvous point in 0.000118 seconds.
18/5/2022 | 16:22:41 -> Thread 3 has reached the rendezvous point in 0.000131 seconds.
18/5/2022 | 16:22:41 -> Thread 4 has reached the rendezvous point in 0.000141 seconds.
18/5/2022 | 16:22:41 -> Thread 4 is advancing to the second part.
18/5/2022 | 16:22:41 -> Thread 2 is advancing to the second part.
18/5/2022 | 16:22:41 -> Thread 1 is advancing to the second part.
18/5/2022 | 16:22:41 -> Thread 3 is advancing to the second part.
18/5/2022 | 16:22:41 -> Thread 2 has finished the second part in 0.000018 seconds.
18/5/2022 | 16:22:41 -> Thread 1 has finished the second part in 0.000023 seconds.
18/5/2022 | 16:22:41 -> Thread 3 has finished the second part in 0.000025 seconds.
18/5/2022 | 16:22:41 -> Thread 4 has finished the second part in 0.000025 seconds.
18/5/2022 | 16:22:41 -> The process has written to the output file. The total time spent is 0.000561 seconds
tgknyhn@tgknyhn:~/Desktop/hw5$
```

n = 3 , m = 8 , dimensions = 8 x 8

```
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
gcc -ggdb3 -Wall source/main.c source/file.c -o hw5 -lpthread -lm && ./hw5 -i "input1.txt" -j "input2.txt" -o "output.csv" -n 3 -m 8
18/5/2022 | 16:25:47 -> Two matrices of size 8x8 have been read. The number of threads is 8
18/5/2022 | 16:25:47 -> Thread 2 has reached the rendezvous point in 0.000096 seconds.
18/5/2022 | 16:25:47 -> Thread 7 has reached the rendezvous point in 0.000142 seconds.
18/5/2022 | 16:25:47 -> Thread 5 has reached the rendezvous point in 0.000133 seconds.
18/5/2022 | 16:25:47 -> Thread 8 has reached the rendezvous point in 0.000135 seconds.
18/5/2022 | 16:25:47 -> Thread 4 has reached the rendezvous point in 0.000169 seconds.
18/5/2022 | 16:25:47 -> Thread 3 has reached the rendezvous point in 0.000152 seconds.
18/5/2022 | 16:25:47 -> Thread 1 has reached the rendezvous point in 0.000101 seconds.
18/5/2022 | 16:25:47 -> Thread 6 has reached the rendezvous point in 0.000159 seconds.
18/5/2022 | 16:25:47 -> Thread 6 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 8 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 1 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 8 has finished the second part in 0.000021 seconds.
18/5/2022 | 16:25:47 -> Thread 1 has finished the second part in 0.000010 seconds.
18/5/2022 | 16:25:47 -> Thread 7 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 5 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 5 has finished the second part in 0.000005 seconds.
18/5/2022 | 16:25:47 -> Thread 7 has finished the second part in 0.000011 seconds.
18/5/2022 | 16:25:47 -> Thread 2 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 6 has finished the second part in 0.000011 seconds.
18/5/2022 | 16:25:47 -> Thread 3 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 2 has finished the second part in 0.000012 seconds.
18/5/2022 | 16:25:47 -> Thread 4 is advancing to the second part.
18/5/2022 | 16:25:47 -> Thread 3 has finished the second part in 0.000000 seconds.
18/5/2022 | 16:25:47 -> Thread 4 has finished the second part in 0.000010 seconds.
18/5/2022 | 16:25:47 -> The process has written to the output file. The total time spent is 0.000647 seconds
tgknyhn@tgknyhn:~/Desktop/hw5$ make run
```