OS Assignment 1, Report

Name : Ahmik Virani

Roll No. : ES22BTECH11001

Low Level Design of The Code

The code is majorly 4 parts: The include files, The global variables, The isTetraHedralFunction and the main function.

The include files are listed below:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <stdbool.h>
```

Next we have global variables:

```
int N, K;
int *array;
```

N and K are as defined in the question.

array is defined globally and will be allocated dynamically in the main function. It will store all values and weather they are tetrahedral or not.

Next part is the isTetrahedralNumber function

```
bool isTetrahedralNumber(int x) {
    for(int i = 1; i<=x; i++)
    {
        if(x == ( i * (i + 1) * (i + 2) / 6) )
        {
            return true;
        }
    }

    return false;

}
```

Here we take an integer input and return true if the number is tetrahedral and false if it is not.

Finally we have the main function

```c
int main(int argc, char const *argv[])
{
    FILE *fileIn = fopen("InFile.txt", "r");        //Open the input file named "InFile.txt" in read only mode

    if(fileIn == NULL)                              //Check if the file in NULL
    {
        perror("error opening file\n");             //If it is NULL, which means it does not exist, throwing an error message
        return 1;                                   //A value more than zero is returned to show that something went wrong in the execution
    }

    /*
    The following two lines scan the values of N and M from the input file
    The first value scanned is N
    The second value scanned is M
    */
    fscanf(fileIn, "%d", &N);                       //value N is scanned as an integer from the input file
    fscanf(fileIn, "%d", &K);                       //value M is scanned as an integer from the input file

    fclose(fileIn);                                 //Once all the input values are taken, we close the input file

    const char *name = "OS";                        //Define the name of the shared memory as OS
    const int SIZE = 4096;                          //Define size of shared memory

    int shm_fileDescriptor;                         //Shared memory file descriptor
    void *ptr;                                      //Pointer to shared memory object

    array = (int *)calloc(N, sizeof(int));          //Dynamically allocating memory to the global variable array

    shm_fileDescriptor = shm_open(name, O_CREAT | O_RDWR, 0666);    //Create the shared memory object
    ftruncate(shm_fileDescriptor, SIZE);                            //Configure the size of the shared memory object

    ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fileDescriptor, 0);     //Memory map to the shared memory object

    /*
    Below is the loop to execute K processes simultaneously
    */
    for(int i= 0; i<K; i++)
    {
        pid_t pid;
        pid = fork();                               //Fork a child process

        if(pid < 0)                                 //Error occured
        {
            fprintf(stderr, "Fork Failed");         //Print an error message
            return 1;
        }
```

```
        else if(pid == 0)                           //Child Process
        {

            int localBuffer[N/K + 1];               //Create a local buffer of size N/K + 1

            for(int j = 0; j< N/K + 1; j++)
            {
                localBuffer[j] = 0;                 //Initialize all the values of the buffer to 0
            }

            int z = 0;                              //This counts the number of tetrahedral numbers in the range this thread calculates

            /*
            The loop below checks each number and stores the tetradehral numbers in the local buffer
            */
            for(int j = i; j < N; j+=K)
            {
                if( isTetrahedralNumber( j + 1 ))
                {
                    localBuffer[z] = j + 1;
                    array[j] = localBuffer[z];
                    z++;
                }
            }

            /*
            Below loop is for writing to the shared memory from the local buffer
            */
            for(int j = 0; j < z ; j++)
            {
                int length = sprintf((char *)ptr, "%d", localBuffer[j]);
                ptr += length;
            }

            char output[50];
            sprintf(output, "OutFile%d.txt", i + 1);
            FILE *fileOut = fopen(output, "w");                 //Open the output file in write mode

            if(fileOut == NULL)                                 //If file does not exist throw an error message
            {
                perror("Error opening file\n");
                return 1;
            }

            /*
            The loop below is for printing to the output file
            */
            for(int j = i; j < N; j+=K)
            {
                if(array[j] != 0)
                {
                    fprintf(fileOut, "%d: a tetrahedral number\n", j+1);
                }

                else
                {
                    fprintf(fileOut, "%d: Not a tetrahedral number\n", j+1);
                }
            }

            fclose(fileOut);                                    //Closing the file
            exit(0);
        }
    }

    shm_unlink(name);                       //Remove the shared memory object


    return 0;
}
```

In the above we do all the file IO as well as processes creation and execution.

How processes are created

First we create the shared memory

```
const char *name = "OS";                        //Define the name of the shared memory as OS
const int SIZE = 4096;                          //Define size of shared memory

int shm_fileDescriptor;                         //Shared memory file descriptor
void *ptr;                                      //Pointer to shared memory object

array = (int *)calloc(N, sizeof(int));          //Dynamically allocating memory to the global variable array

shm_fileDescriptor = shm_open(name, O_CREAT | O_RDWR, 0666);    //Create the shared memory object
ftruncate(shm_fileDescriptor, SIZE);                            //Configure the size of the shared memory object

ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fileDescriptor, 0);     //Memory map to the shared memory object
```

We then run a loop K times to create K child processes

```
for(int i= 0; i<K; i++)
{
    pid_t pid;
    pid = fork();                                   //Fork a child process

    if(pid < 0)                                     //Error occured
```

Each child has a local buffer of size N/K + 1, note the size is N/K + 1 and not N/K, because say there are 10 numbers and 3 processes, C interprets integer value of N/K as 3 and not 4, but we know 1 of the 3 processes needs to check 4 numbers and so N/K + 1.

```
int localBuffer[N/K + 1];               //Create a local buffer of size N/K + 1

for(int j = 0; j< N/K + 1; j++)
{
    localBuffer[j] = 0;                 //Initialize all the values of the buffer to 0
}
```

After all the local buffers and processes are created, each process check numbers in a modular fashion, example there are 10 numbers and 3 processes

Process 1 checks : 1, 4, 7, 10

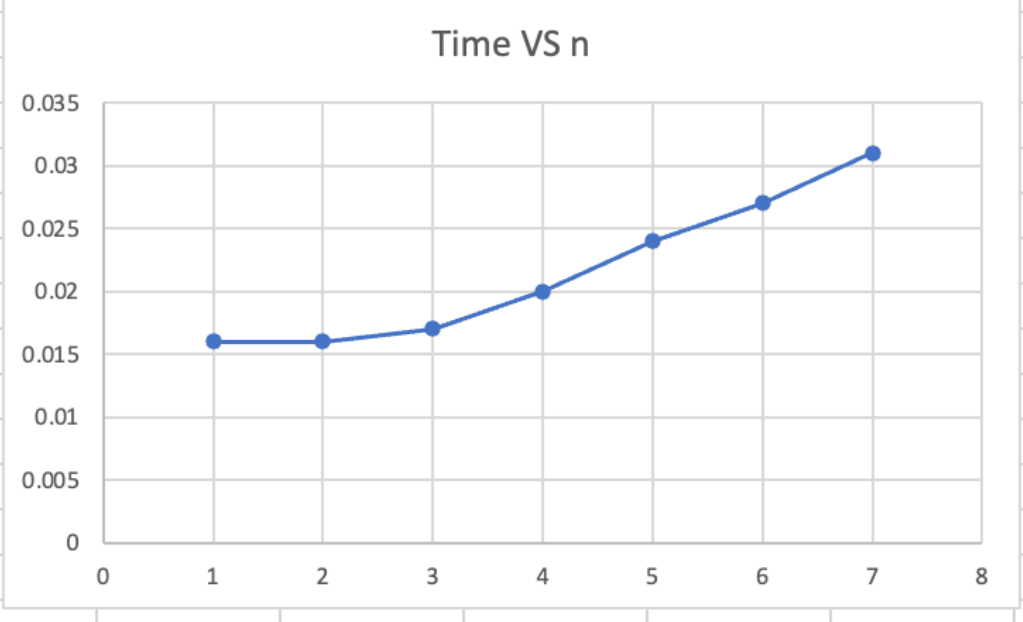
Process 2 checks : 2, 5, 8

Process 3 checks : 3, 6, 9

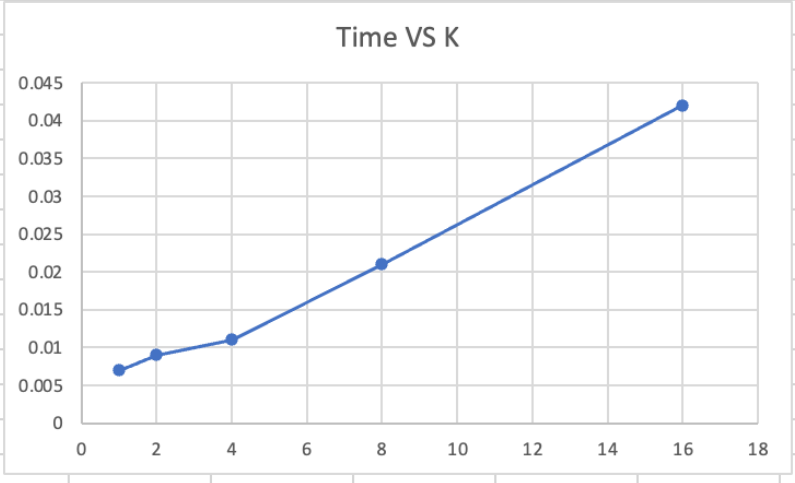
```
for(int j = i; j < N; j+=K)
{
    if( isTetrahedralNumber( j + 1 ))
    {
        localBuffer[z] = j + 1;
        array[j] = localBuffer[z];
        z++;
    }
}
```

Analysis of the code

n	time	k
1	0.016	
2	0.016	
3	0.017	
4	0.02	
5	0.024	
6	0.027	
7	0.031	



k	time
1	0.007
2	0.009
4	0.011
8	0.021
16	0.042



Clearly time increases as n increases as well as k increases.

As n increases size N = 2 ^ (3 * n) increases, and so work done per process increases and so time increases

As for the reason for K, maybe in my laptop there is more throughput generated while multiprocessing and so the time is increasing, but generally multiprocessing decreases the time.