Faculty of Engineering & Technology
Department of Electrical & Computer Engineering
ENCS3390: Operating System Concepts
Second Semester, 2023/2024

Project 1 Report

Name: Taleen Bayatneh
Student ID: 1211305

Instructor: Abdel Salam Sayyad
Section: 2
Date: 4/5/2024

1. **Abstract**:

This project aims to develop and analyze a program that calculates the average Body Mass Index (BMI) from a given dataset. we will implement the program by using three different approaches: a naive approach without any parallel processing, a multiprocessing approach using multiple child processes, and a multithreading approach using multiple threads. also, we will measure and compare the performance of these approaches in terms of execution time.

Table of Contents

Table of figures

List of tables

2. Introduction:

2.1 Naive approach:

Naive is a simple and inefficient way to see wherever one string occurs within another is to examine every place it could be at, one by one, to examine if it's there

(involve simple arithmetic operations without considering potential issues such as input validation, error handling, or edge cases.)

2.2 Multiprocessing:

Multiprocessing is the utilization of two or more central processing units (CPUs) in a single computer system. Its definition can vary depending on the context, but generally it refers to a system's ability to support multiple CPUs and its capacity to distribute work among them.

Multicore processors today are easily capable of having 12, 24 or even more microprocessor cores on the same motherboard, enabling the effective and concurrent processing of numerous tasks.

(a programming technique that involves executing multiple processes concurrently to improve the performance and efficiency of a computer system, particularly on multi-core or multi-processor architectures.)

2.3 Multithreading:

Multithreading is a CPU feature that allows two or more instruction threads to execute independently while sharing the same process resources. A thread is a self-contained sequence of instructions that can execute in parallel with other threads that are part of the same root process.

(a programming technique that involves executing multiple threads concurrently within the same process. A thread is the smallest unit of execution within a process, and multithreading allows multiple threads to execute independently, potentially improving the performance and responsiveness of a program.)

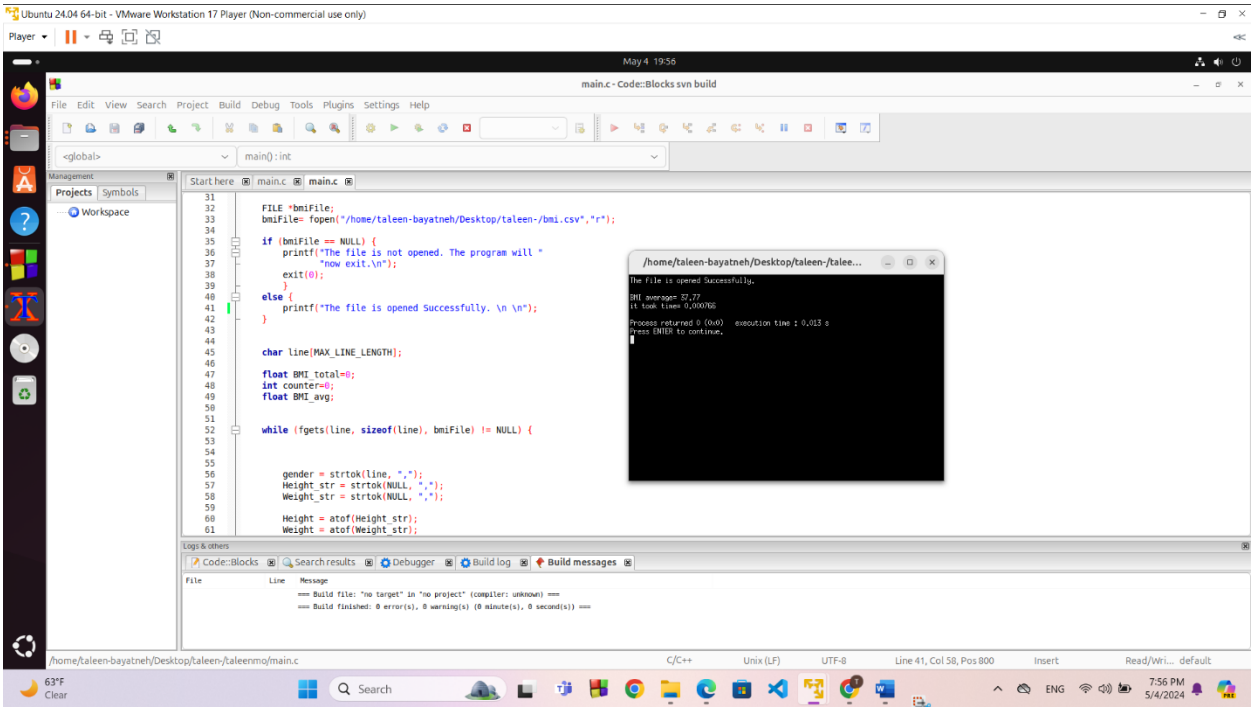
3. Procedure

3.1 Naive approach:

3.1.1 Code:

```c
//taleen bayatneh 1211305 sec: 2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX_LINE_LENGTH 1000

float BMI_naive_appraoch(float Height, float Weight)
{
    float BMI;
    float hight_m= Height / 100.0;

    BMI = Weight / (hight_m * hight_m);

    return BMI;
}

int main()
{
struct  timespec begin;
    timespec_get(&begin,TIME_UTC);

    char *gender;
    char *Height_str;
    char *Weight_str;

    float Height;
    float Weight;
    float BMI;


      FILE *bmiFile;
      bmiFile= fopen("/home/taleen-bayatneh/Desktop/taleen-/bmi.csv","r");

    if (bmiFile == NULL) {
        printf("The file is not opened. The program will "
               "now exit.\n");
        exit(0);
        }
    else {
        printf("The file is opened Successfully. \n \n");
    }


    char line[MAX_LINE_LENGTH];

    float BMI_total=0;
    int counter=0;
    float BMI_avg;


    while (fgets(line, sizeof(line), bmiFile) != NULL) {



        gender = strtok(line, ",");
        Height_str = strtok(NULL, ",");
        Weight_str = strtok(NULL, ",");

        Height = atof(Height_str);
        Weight = atof(Weight_str);

      // Calculate BMI
      if (Height != 0 && Weight != 0){
       BMI= BMI_naive_appraoch(Height, Weight);
      BMI_total+= BMI;
      counter++;
      }


    }
    BMI_avg= BMI_total / counter;

    printf("BMI average= %.2lf\n", BMI_avg);
      fclose(bmiFile);


    struct  timespec end;
     timespec_get(&end,TIME_UTC);
     double time_spent=(end.tv_sec - begin.tv_sec)+(end.tv_nsec - begin.tv_nsec)/1000000000.0;
     printf("it took time= %lf\n",time_spent);

    return 0;
}
```

3.1.2 the execution time for all the code :



3.1.3 Discussion:

In naive approach I made a function to calculate BMI for each line by entering the height and the weight to the function , and then in the main function I opened the file by fopen() and then get the values by dividing the lines by (,) using strtok(),after that I called the naïve function to the calculate in while loop , at the end I divide the total of BMI by there number.

the execution time = 0.000766 , so the performance = 1305.48303 .

3.2 Multiprocessing approach:

3.2.1 Code:

3.2.1.1 Pipe and child and parent in main :

```
    int num_of_child = 5;
    float avg=0;
    float total=0;
    read_data();
    double child_lines =0.0;
    child_lines= num_people/num_of_child;
    ceil(child_lines);

    int fd[num_of_child][2];

    for (int x =0; x<num_of_child; x++)
    {
        if (pipe(fd[x]) == -1)
        {
            printf("error with opening the pipe \n");
            return 1;
        }

    }

// Loop to create child processes
    for (int y = 1; y <= num_of_child; y++)
    {
        int pid = fork();
        if (pid == -1)
        {
            printf("Fork failed\n");
        }
        else if (pid == 0) // Child process
        {

            total += child(child_lines, y);
            write(fd[y-1][1], &total, sizeof(float));
            close(fd[y-1][1]);
            exit(0);
        }
        else // Parent process
        {
            close(fd[y-1][1]);
            read(fd[y-1][0], &total, sizeof(float));
            close(fd[y-1][0]);
        }
    }
```

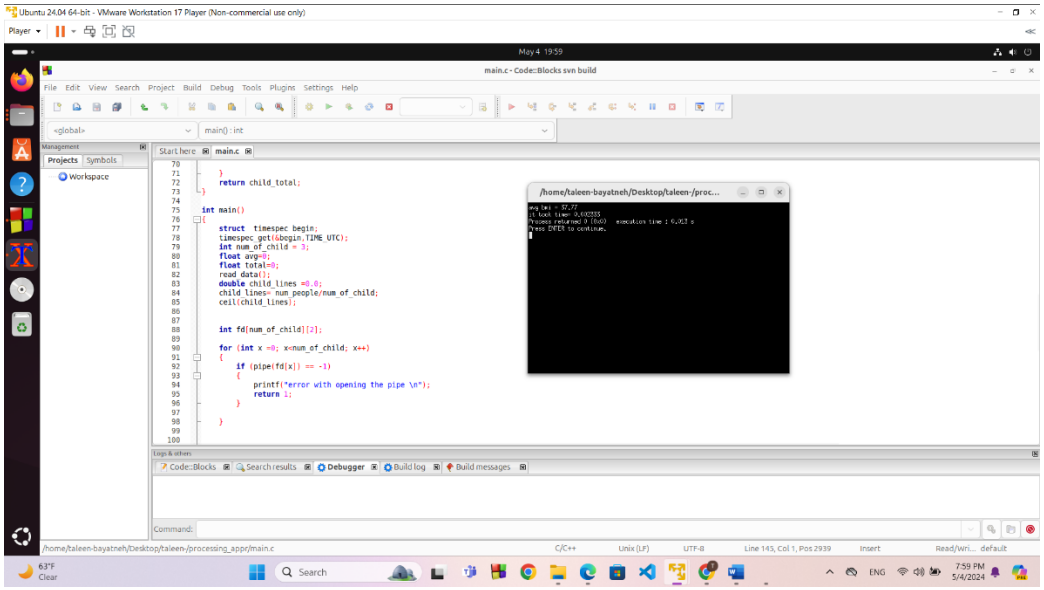
3.2.1.2 Child process:

```
float child(double child_lines,int numofchild)
{
    float child_total =0;
    int x = child_lines*(numofchild-1) ;

    for(int i=x+1; i<=child_lines+x; i++)
    {
        if (i>=num_people)
        break;
        child_total += (BMI_naive_appraoch(people[i].height,  people[i].weight));

    }
    return child_total;
}
```

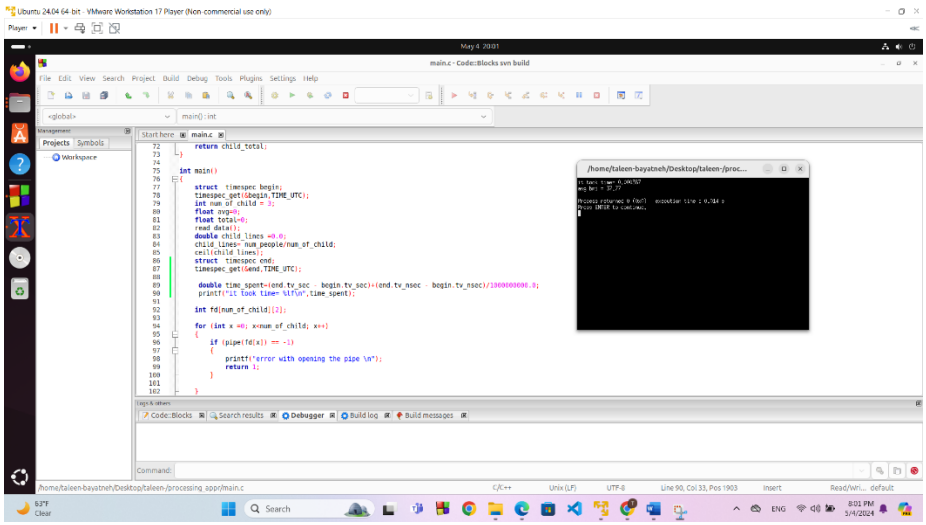
Number of children	Execution time	Performance
2	0.001809	552.7915
3	0.002280	438.5964
4	0.003001	333.2222
5	0.003416	292.7400

Table 1:multiprocessing execution time with different number of child

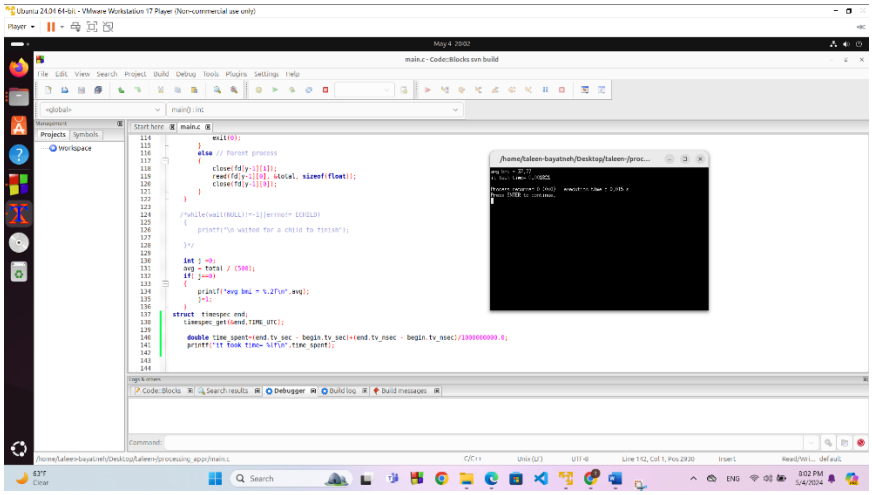
3.2.2 the execution time :



All the code



Reading the data



For parallel process

3.2.3 Discussion:

In this code I open the file in a function and then call it in the main function , also int the main I made the pipe in for loop because I need more than one child ,also we have a loop for fork() function depend on number of children and I call the child function when the pid = 0 which mean the child ,also I used the pipe to connect the children and parent by using read and write .

The child function will return the total of BMI for a number lines that will takes ,also it takes the number of child that it will enter ,inside the function we have for loop it starts from the line according to the number of child .

The execution for all code=0.00233 ,for reading from file=0.000367,for parallel=0.001821

3.3 Multithreading approach (joinable threads):

3.3.1 Code:

3.3.1.1 thread function:

```
65  void* threadFunction (void* thread_number)
66  {
67
68      int num= *(int*)thread_number;
69      int x = thread_lines*(num) ;
70      float sub_total=0;
71      pthread_mutex_lock(&lock);
72
73      for(int i=x+1; i<=thread_lines+x ; i++)
74      {
75          if (i>=num_people)
76              break;
77
78          //printf("Height:%.2f \t Weight:%.2f \n", people[i].height, people[i].weight);
79
80          sub_total += (BMI_naive_appraoch(people[i].height, people[i].weight));
81
82
83      }
84      th ++;
85      //printf("sub_total= %.2f ,T=%d\n", sub_total,num);
86      final_total+=sub_total;
87      pthread_mutex_unlock(&lock);
88
89      free(thread_number);
90      if (th==numOfThds)
91          printf("bmi avg = %.2lf\n", final_total/ (num_people-1));
92
93
94  }
```

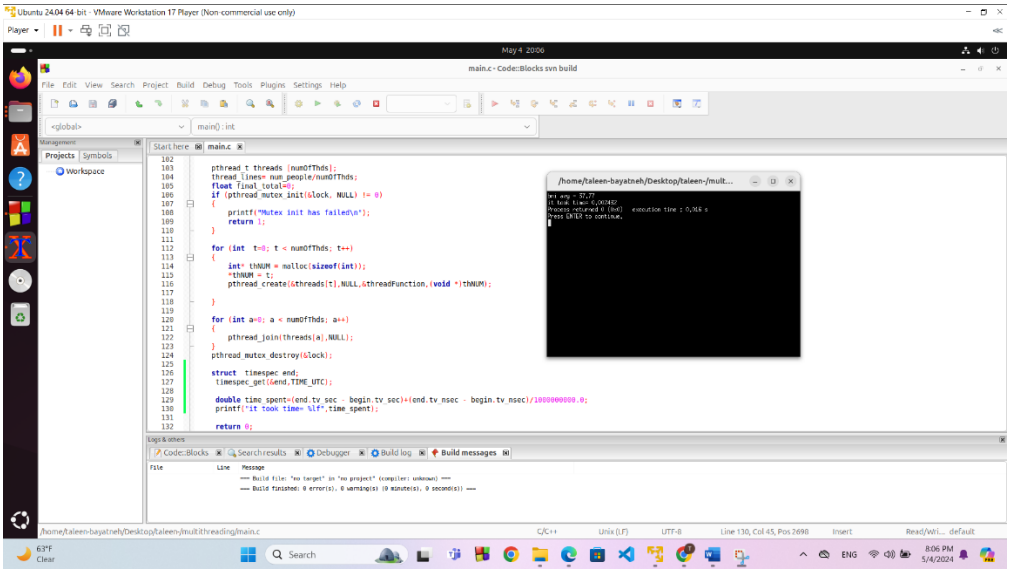
3.3.1.2 thread create and join in main function :

```
104     pthread_t threads [numOfThds];
105     thread_lines= num_people/numOfThds;
106     float final_total=0;
107     if (pthread_mutex_init(&lock, NULL) != 0)
108     {
109         printf("Mutex init has failed\n");
110         return 1;
111     }
112
113
114     for (int  t=0; t < numOfThds; t++)
115     {
116         int* thNUM = malloc(sizeof(int));
117         *thNUM = t;
118         pthread_create(&threads[t],NULL,&threadFunction,(void *)thNUM);
119
120     }
121
122     for (int a=0; a < numOfThds; a++)
123     {
124         pthread_join(threads[a],NULL);
125     }
126
127
128     pthread_mutex_destroy(&lock);
129
130  struct  timespec end;
131      timespec_get(&end,TIME_UTC);
132
133      double time_spent=(end.tv_sec - begin.tv_sec)+(end.tv_nsec - begin.tv_nsec)/1000000000.0;
134      printf("it took time= %lf\n",time_spent);
135
136      return 0;
137  }
138
```

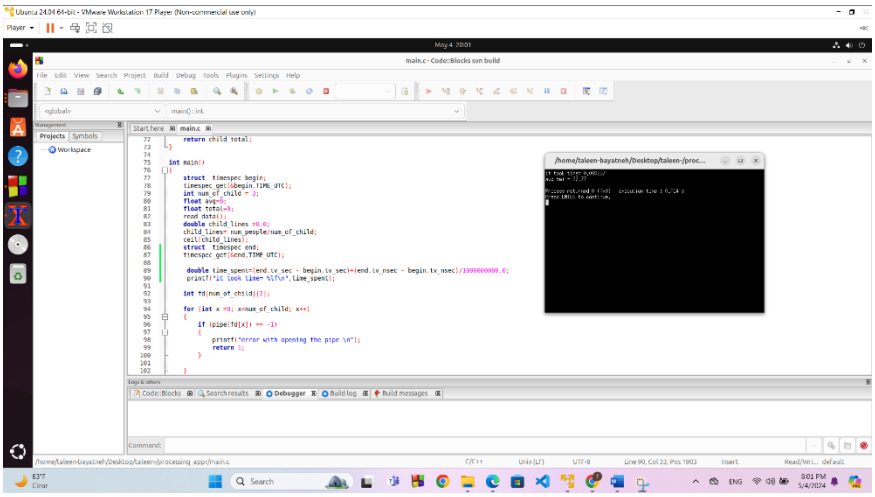
Number of threads	Execution time	Performance
2	0.003613	276.7783
3	0.001483	674.3088
4	0.001645	607.9027
5	0.002585	386.8471

Table 2 : multithreading execution time with different number of threads

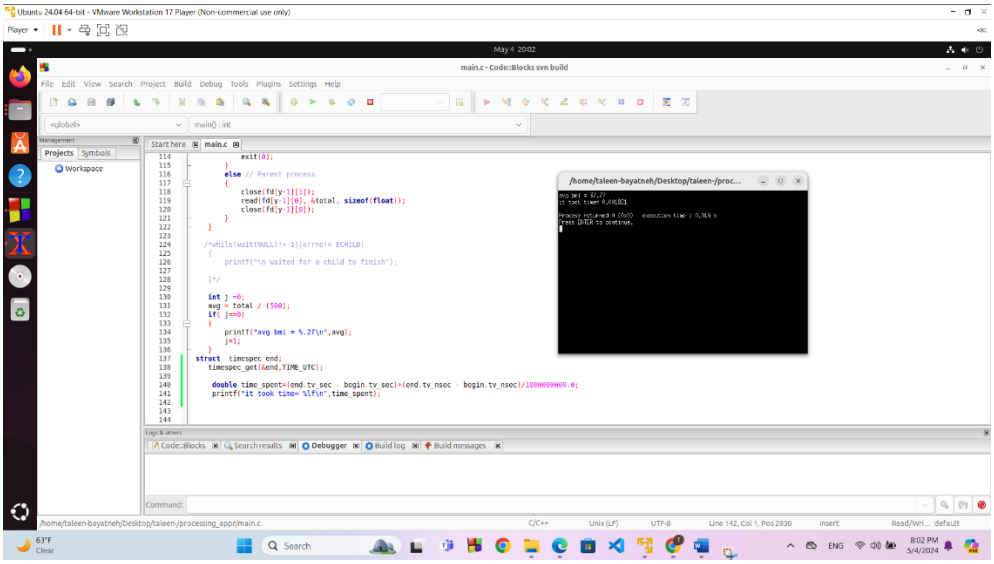
3.3.2 the execution time :



All the code execution



Reading the data



parallel

3.3.3 Discussion:

In the main function we create the pthread in for loop it depends on how many thread that I have to create , I use the joinable thread in loop to make it parallel , also Initialized the mutex in order to get a valid number.in the thread function I used the mutex before the for loop and unlocked it before the print of avg (result) ,inside the loop I called a function to calculate the BMI for each line and added to sub total that will go to the final total .

The execution for all code=0.002432 ,for reading from file=0.000362,for parallel=0.001232

4. Measurements and discussion

Approach	Execution time	Performance
Naive	**0.000766**	**1305.48303**
Multiprocessor 2 processor	**0.001809**	**552.7915**
Multiprocessor 3 processor	**0.002280**	**438.5964**
Multiprocessor 4 processor	**0.003001**	**333.2222**
Multiprocessor 5 processor	**0.003416**	**292.7400**
Multithreading 2 threads	**0.003613**	**276.7783**
Multithreading 3 threads	**0.001483**	**674.3088**
Multithreading 4 threads	**0.001645**	**607.9027**
Multithreading 5 threads	**0.002585**	**386.8471**

Table 2: table to compare between the three approaches

- **percentage off the serial part of 5 threads:**

Time read from file = 0.000362 sec

Time total BMI and average =0.001232 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.000362 +0.001232) = 0.1594%sec

Portion= percentage off the serial part /total time

Portion= 0.1594%/0.002585=61.6%

Speed up = 1/(S+((1-S)/N)) = 1 / (0.616+(0.384 /5))=1.4

- **percentage off the serial part of 4 threads:**

Time read from file = 0.000362 sec

Time total BMI and average =0.001232 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.000362 +0.001232) = 0.1594%sec

Portion= percentage off the serial part /total time

Portion= 0.1594%/**0.001645**=36.1%

Speed up = 1/(S+((1-S)/N)) = 1 / (0.361+(0.639 /4))=1.9

- **percentage off the serial part of 3threads:**

Time read from file = 0.000362 sec

Time total BMI and average =0.001232 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.000362 +0.001232) = 0.1594%sec

Portion= percentage off the serial part /total time

Portion= 0.1594%/0.001483=107%

Speed up = 1/(S+((1-S)/N)) = 1 / (0.361+(0.384 /3))=0.95

- **percentage off the serial part of 2threads:**

Time read from file = 0.000362 sec

Time total BMI and average =0.001232 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.000362 +0.001232) = 0.1594%sec

Portion= percentage off the serial part /total time

Portion= 0.1594%/0.003613=44.11%

Speed up = 1/(S+((1-S)/N)) = 1 / (0.441+(0.384 /2))=1.3

- **percentage off the serial part of 5 children:**

Time read from file = 0.00233 sec

Time total BMI and average =0.001821 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.00233 +0.001821) = 0.4151%sec

Portion= percentage off the serial part /total time

Portion= 0.4151%/**0.003416**=121.5%

Speed up = 1/(S+((1-S)/N)) = 1 / (01.215+(0.384 /5))=0.85

- **percentage off the serial part of 4 children:**

Time read from file = 0.00233 sec

Time total BMI and average =0.001821 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.00233 +0.001821) = 0.4151%sec

Portion= percentage off the serial part /total time

Portion= 0.4151%/**0.003001**= 138.3%

Speed up = 1/(S+((1-S)/N)) = 1 / (01.383+(-0.383 /4))=0.77

- **percentage off the serial part of 3 children:**

Time read from file = 0.00233 sec

Time total BMI and average =0.001821 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.00233 +0.001821) = 0.4151%sec

Portion= percentage off the serial part /total time

Portion= 0.4151%/0.002280= 182.02%

Speed up = 1/(S+((1-S)/N)) = 1 / (01.383+(-0.82 /3))=0.90

- **percentage off the serial part of 2 children:**

Time read from file = 0.00233 sec

Time total BMI and average =0.001821 sec

serial percentage = 100% *(Time read from file + Time total BMI and average)

=100% *(0.00233 +0.001821) = 0.4151%sec

Portion= percentage off the serial part /total time

Portion= 0.4151%/0.001809=229.4 %

Speed up = 1/(S+((1-S)/N)) = 1 / (02.294+(-2.28 /2))=0.866

5 conclusion :

-According to the result the maximum speed is when the number of thread is 4 with a number of core =4
-According to the result the optimal number of child processes is 4 with a number of core =4

-According to the result the maximum speed is when the number of child is 3 with a number of core =4
-According to the result the optimal number of child processes is 3 with a number of core =4

5. References:

- https://www.tutorialspoint.com/c-program-for-naive-algorithm-for-pattern-searching#:~:text=Naive%20is%20a%20simple%20and,size%20of%20the%20container%20string.
- https://www.techtarget.com/searchdatacenter/definition/multiprocessing
- https://www.techopedia.com/definition/24297/multithreading-computer-architecture