Academic Year	**2024**
Semester	☐ Fall ☒ Winter ☐ Summer
Course Code - Name	CSCI 3310 – System Programming
Instructor	Dr. Razi Iqbal
Assessment	Assignment 1

Instructions:

In order to obtain maximum marks in this assignment, please ensure the followings:

- Submit this assignment by writing your solution in this document under the Solution heading below. Do not use a separate document.
- Copy and paste the code from your editor in this document for both the parts.
- Make sure to take a screenshot of the output of both the parts.
- **Make sure you take the screenshot of the Activity Monitor for Requirement 1 and 2.**
- **For Requirement 3, make sure to provide a concrete reason in plain English.**
- This assignment has a weightage of **10%** marks of the course.
- This is **NOT** a group assignment so **students having similar assignments will get a 0.**
- The assignment deadline is **midnight Feb. 11, 2024**. Submissions after the deadline will not be accepted.

Question 1

Threads in modern operating systems are used as a mean of improving the efficiency of the programs and avoiding lags in performance. Most of the modern operating systems provide threading using Thread Libraries. Some programming languages as Java also provide libraries for managing threads that can be incorporated in to programs to enhance the performance of the programs.

In this assessment, you are required to demonstrate the use of threading and provide an evidence of performance boost of the program using the built-in threading libraries. The assessment is two-folds. Below are the requirements of the assessment:

Part 1:
You are required to write a program in C that takes (1,000,000,000) as an input (you are free to take this input from command line arguments or from the user). Once the input is taken, there should be 3 threads in your program performing the following operations:

1. Child Thread 1:
    a. Displays its ID
    b. Find incremental sum of all the numbers until and including 1,000,000,000
    c. Exit itself once completed the operation

2. Child Thread 2:
    a. Displays its ID
    b. Find incremental sum of all the even numbers until and including 1,000,000,000
    c. Exit itself once completed the operation

3. Parent / Main Thread:
    a. Displays its ID
    b. Creates Child Threads 1 and 2
    c. Merges Child Thread 1 and 2
    d. Displays the sum of all the numbers returned by Child Thread 1
    e. Displays the sum of all the numbers returned by Child Thread 2
    f. Calculates the following:
        i. sum of all the numbers returned by Child Thread 1 / sum of all the numbers returned by Child Thread 2
    g. Displays the result of above calculation

Requirement 1:
Analyze the time taken by the program to complete, e.g., review the usage of cores using Activity Monitor program in Ubuntu and find a way to monitor the time taken by the program to complete its execution. Furthermore, make sure, the calculations made by the program are correct and accurate. Do NOT forget to take a screenshot of the Activity Monitor.

Part 2:
You are required to write a program in C that takes (1,000,000,000) as an input (you are free to take this input from command line arguments or from the user). Once the input is taken, the program should perform the following operations:

1. Find incremental sum of all the numbers until and including 1,000,000,000
2. Find incremental sum of all the even numbers until and including 1,000,000,000

3. Display the sum of all the numbers until and including 1,000,000,000
4. Displays the sum of all the even numbers until and including 1,000,000,000
5. Calculates the following:
    - sum of all the numbers in Step 1 / sum of all the numbers in Step 2
6. Displays the result of above calculation

Requirement 2:
Analyze the time taken by the program to complete, e.g., review the usage of cores using Activity Monitor program in Ubuntu and find a way to monitor the time taken by the program to complete its execution. Do NOT forget to take a screenshot of the Activity Monitor.

Requirement 3:
Compare the results obtained by the analysis of Requirement 1 and Requirement 2. Make sure to write down the reasons of your analysis in plain English.

Solution

Part 1.

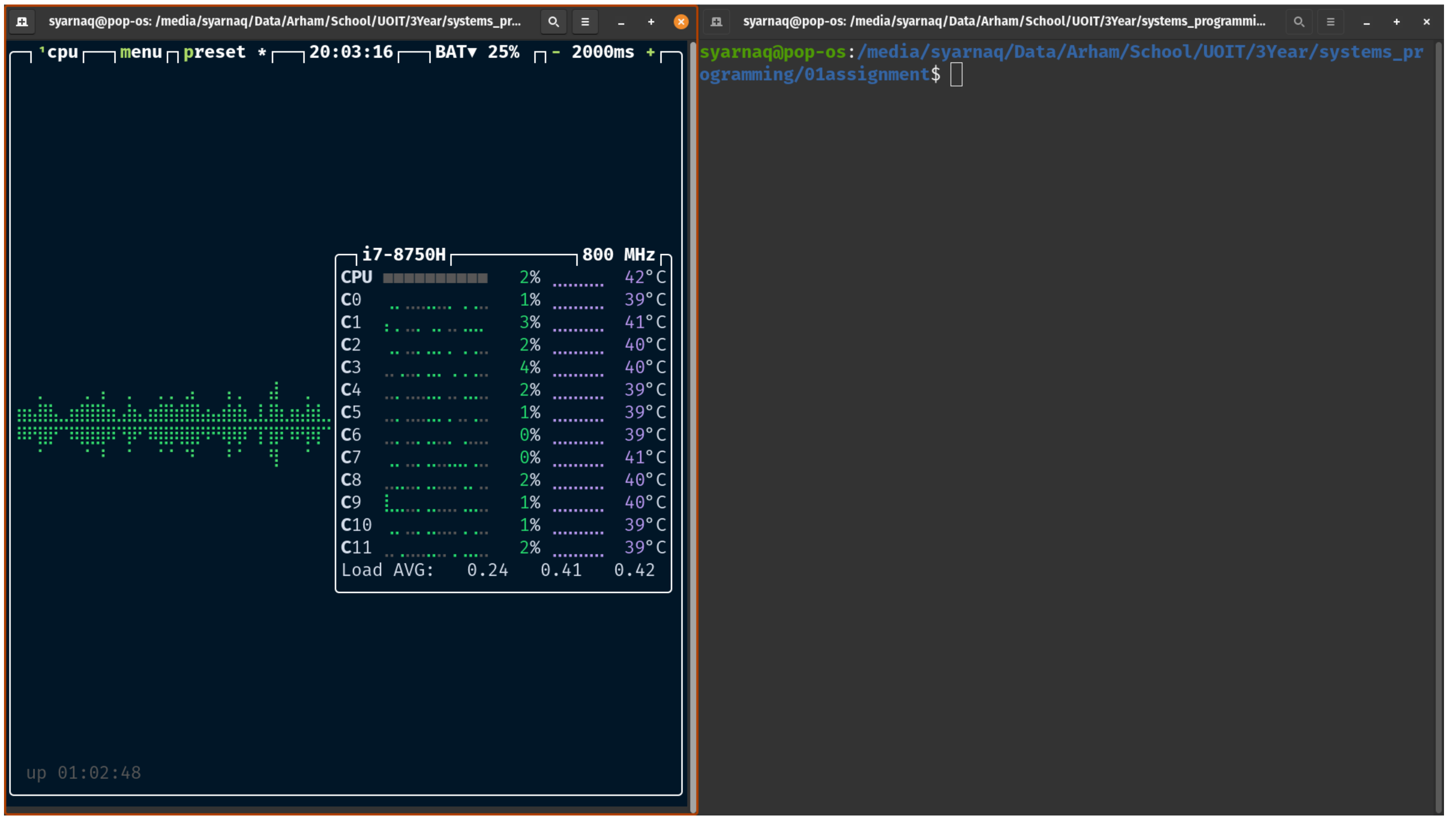
Code:

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

void* incremental_sum(void* arg){

__intptr_t* num = (__intptr_t*)arg;
// Displaying thread ID and purpose
printf("Child Thread 1 - ID: %lu\n", pthread_self());
printf("Calculating incremental sum of numbers from 1 to %ld\n\n", *num);

// Calculating and returning incremental sum of passed value
__intptr_t inc_sum = 0;

for (__intptr_t i = 1; i < *num + 1; i++)
{
inc_sum += i;
}
return (void*) inc_sum;

}

void* incremental_even_sum(void* arg){

__intptr_t* num = (__intptr_t*)arg;
// Displaying thread ID and purpose
printf("Child Thread 2 - ID: %lu\n", pthread_self());
printf("Calculating incremental sum of even numbers from 1 to %ld\n\n", *num);

// Calculating and returning incremental sum of even numbers up to passed value
(inclusive)
__intptr_t inc_sum = 0;

for (__intptr_t i = 2; i < *num + 1; i+=2)
{
inc_sum += i;
}
return (void*) inc_sum;

}
```

```
int main(int argc, char const *argv[])
{
// used to calculate execution time in terms of clock ticks
clock_t start, end;
double time_elapsed;

// recording start time
start = clock();

// Displaying thread ID and purpose
printf("\nParent thread - ID: %lu\n", pthread_self());
printf("Creating child threads for calculating incremental sums...\n");
printf("Calculating ratio...\n\n");

// storing command line argument
__intptr_t value = atoi(argv[1]);

// will store child ids
pthread_t child1;
pthread_t child2;

// will store child results
void* inc_sum;
void* inc_sum_even;
// creating child threads
pthread_create(&child1, NULL, &incremental_sum, (void*) &value);
pthread_create(&child2, NULL, &incremental_even_sum, (void*) &value);

// joining child threads
pthread_join(child1, &inc_sum);
pthread_join(child2, &inc_sum_even);

// calculating ratio
double ratio = (double)(__intptr_t)inc_sum / (__intptr_t)inc_sum_even;
// displaying results of final calculation
printf("incremental sum = %ld\n", (__intptr_t)inc_sum);
printf("incremental even sum = %ld\n\n", (__intptr_t)inc_sum_even);
printf("incremental sum / incremental even sum = %f\n\n", ratio);

// recording end time
end = clock();
// calculating total elapsed time
time_elapsed = (double)(end - start) / CLOCKS_PER_SEC;
// displaying elapsed time
printf("Execution time = %fs\n", time_elapsed);

return 0;
}
```

Screenshots:

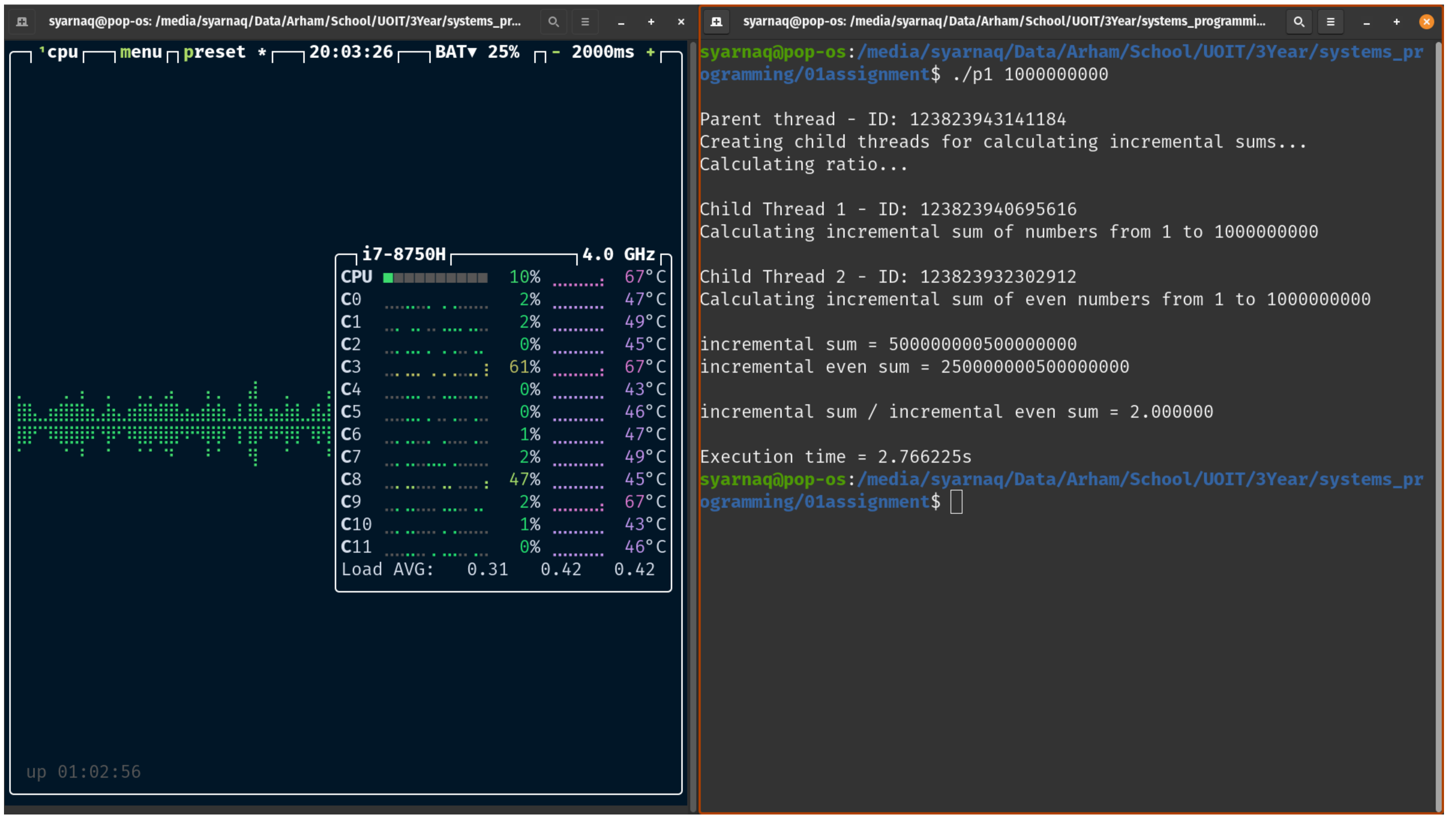
Pre-Execution



```
syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_pr...
¹cpu  menu  preset *  20:03:16  BAT▼ 25%  - 2000ms +

                 i7-8750H                800 MHz
                 CPU               2%      42°C
                 C0                1%      39°C
                 C1                3%      41°C
                 C2                2%      40°C
                 C3                4%      40°C
                 C4                2%      39°C
                 C5                1%      39°C
                 C6                0%      39°C
                 C7                0%      41°C
                 C8                2%      40°C
                 C9                1%      40°C
                 C10               1%      39°C
                 C11               2%      39°C
                 Load AVG:   0.24   0.41   0.42

up 01:02:48
```

```
syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_programmi...
syarnaq@pop-os:/media/syarnaq/Data/Arham/School/UOIT/3Year/systems_programming/01assignment$
```

Post-Execution


syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_pr...
syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_programmi...

```
┌ ¹cpu ┌─┐menu┌┐preset *┌─┐20:03:26┌─┐BAT▾ 25% ┌┐- 2000ms +┌

                              ┌i7-8750H┌─────────┐4.0 GHz┌
                              CPU ■■■■■■■■■■■  10%  67°C
                              C0                 2%  47°C
                              C1                 2%  49°C
                              C2                 0%  45°C
                              C3                61%  67°C
                              C4                 0%  43°C
                              C5                 0%  46°C
                              C6                 1%  47°C
                              C7                 2%  49°C
                              C8                47%  45°C
                              C9                 2%  67°C
                              C10                1%  43°C
                              C11                0%  46°C
                              Load AVG:   0.31   0.42   0.42

 up 01:02:56
```

```
syarnaq@pop-os:/media/syarnaq/Data/Arham/School/UOIT/3Year/systems_pr
ogramming/01assignment$ ./p1 1000000000

Parent thread - ID: 123823943141184
Creating child threads for calculating incremental sums...
Calculating ratio...

Child Thread 1 - ID: 123823940695616
Calculating incremental sum of numbers from 1 to 1000000000

Child Thread 2 - ID: 123823932302912
Calculating incremental sum of even numbers from 1 to 1000000000

incremental sum = 500000000500000000
incremental even sum = 250000000500000000

incremental sum / incremental even sum = 2.000000

Execution time = 2.766225s
syarnaq@pop-os:/media/syarnaq/Data/Arham/School/UOIT/3Year/systems_pr
ogramming/01assignment$ 
```

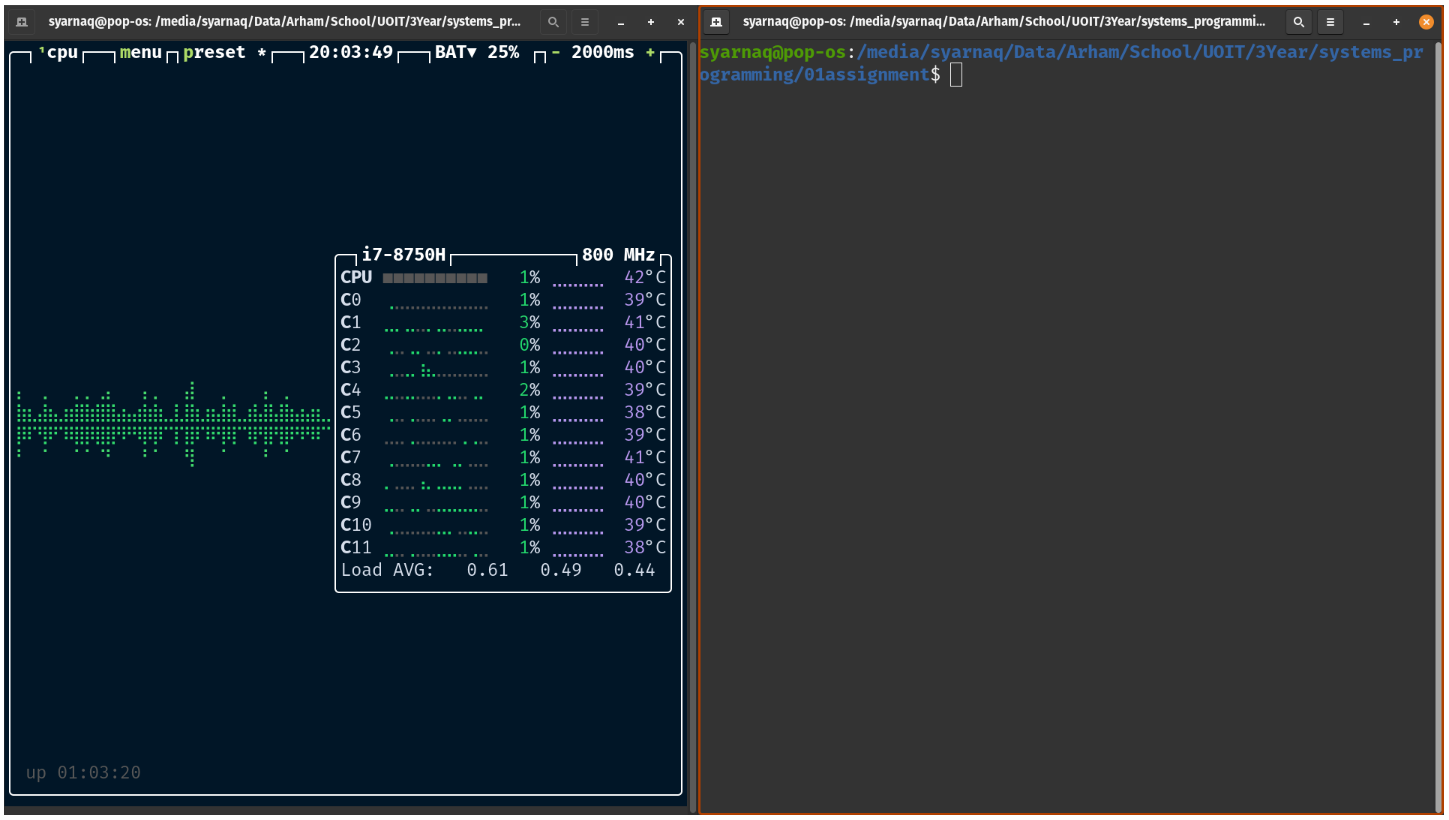
Part 2.

Code:

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


int main(int argc, char const *argv[])
{
// used to calculate execution time in terms of clock ticks
clock_t start, end;
double time_elapsed;
// recording start time
start = clock();


// extract command line argument and convert to int
size_t num = atoi(argv[1]);


// calculating incremental sum
size_t inc_sum = 0;
for (size_t i = 1; i < num + 1; i++)
{
inc_sum += i;
}
// calculating incremental sum of even numbers
size_t inc_sum_even = 0;
for (size_t i = 0; i < num + 1; i+=2)
{
inc_sum_even += i;
}


// calculating ratio
double ratio = (double) inc_sum / inc_sum_even;


// displaying results of final calculation
printf("\nincremental sum = %ld\n\n", inc_sum);
printf("incremental even sum = %ld\n\n", inc_sum_even);
printf("incremental sum / incremental even sum = %f\n\n", ratio);


// recording end time
end = clock();
// calculating total elapsed time
time_elapsed = (double)(end - start) / CLOCKS_PER_SEC;


// displaying elapsed time
printf("Execution time = %fs\n\n", time_elapsed);


return 0;
}
```

Screenshots:

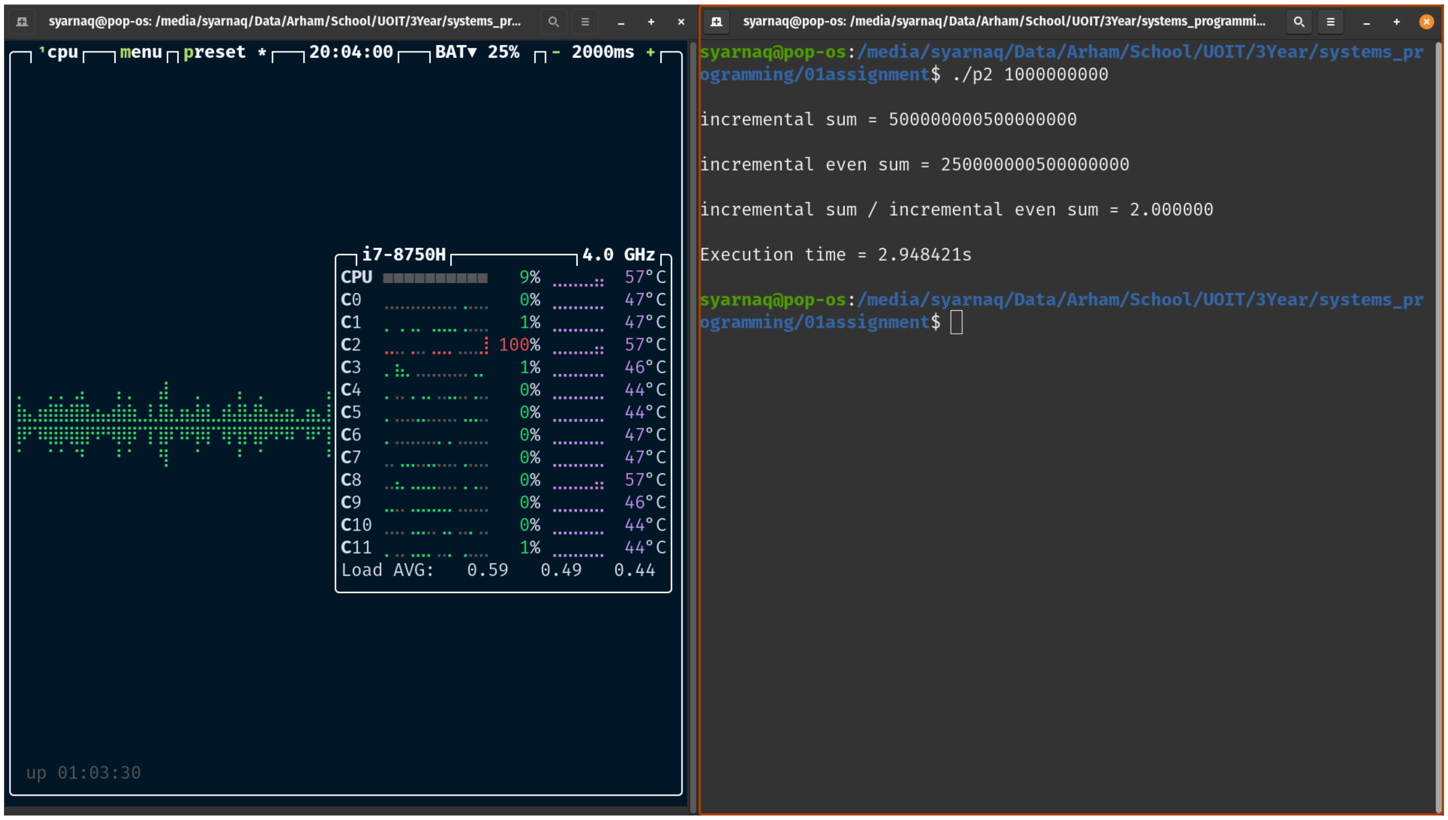
Pre-Execution:



```
syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_pr...

¹cpu  menu  preset *  20:03:49  BAT▾ 25%  - 2000ms +

                    i7-8750H               800 MHz
                    CPU                 1%         42°C
                    C0                  1%         39°C
                    C1                  3%         41°C
                    C2                  0%         40°C
                    C3                  1%         40°C
                    C4                  2%         39°C
                    C5                  1%         38°C
                    C6                  1%         39°C
                    C7                  1%         41°C
                    C8                  1%         40°C
                    C9                  1%         40°C
                    C10                 1%         39°C
                    C11                 1%         38°C
                    Load AVG:   0.61   0.49   0.44

up 01:03:20
```

```
syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_programmi...

syarnaq@pop-os:/media/syarnaq/Data/Arham/School/UOIT/3Year/systems_pr
ogramming/01assignment$ 
```

Post-Execution:


syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_pr...
syarnaq@pop-os: /media/syarnaq/Data/Arham/School/UOIT/3Year/systems_programmi...
cpu
menu
preset *
20:04:00
BAT▾ 25%
- 2000ms +
up 01:03:30

```
i7-8750H              4.0 GHz
CPU          9%    57°C
C0           0%    47°C
C1           1%    47°C
C2         100%    57°C
C3           1%    46°C
C4           0%    44°C
C5           0%    44°C
C6           0%    47°C
C7           0%    47°C
C8           0%    57°C
C9           0%    46°C
C10          0%    44°C
C11          1%    44°C
Load AVG:   0.59   0.49   0.44
```

```
syarnaq@pop-os:/media/syarnaq/Data/Arham/School/UOIT/3Year/systems_programming/01assignment$ ./p2 1000000000

incremental sum = 500000000500000000

incremental even sum = 250000000500000000

incremental sum / incremental even sum = 2.000000

Execution time = 2.948421s

syarnaq@pop-os:/media/syarnaq/Data/Arham/School/UOIT/3Year/systems_programming/01assignment$ 
```

Part 3.

The programs in parts 1 and 2 both accomplish the same task but part 1 is implemented as a multithreaded process while part 2 is implemented as a single threaded process.

The program in part 1 demonstrates a multicore programming approach that leverages true parallelism. This is because each thread in the multithreaded process can be managed independently by the scheduler and so can be assigned a different core(processor). This allows for each thread to be executed simultaneously, distributing processing load across multiple CPU cores and reducing execution times compared to sequential execution. These findings are evident in the screenshots above for part 1 where we can see a large spike in the load of precisely 2 cpu cores (one or each thread) while the program is running.

The program in part 2 is a single threaded process. Since there is only one thread, it can only be processed by a single processor at a time. This means that one processor would be placed under much greater load and program execution times would increase since all tasks would need to be completed sequentially. These facts are evident from the screenshots in part 2 where not only do we observe an increase in execution time as compared to the program in part 1, but we can see that a single core reaches 100% load.

Thus we have successfully showcased how multithreading can be used to reduce execution times and reduce processor load.