OPERATING SYSTEMS

G. HRUTHIK
COE18B023

LAB MID SEM

**Q) Develop a C program that generates the histogram (frequency count
of unique characters in the input text) using multiple threads (thread numbers to match with unique character count) , each thread performing the
counting for that respective character. Compare the efficiency of the threaded version over its equivalent serial version.**

sol)

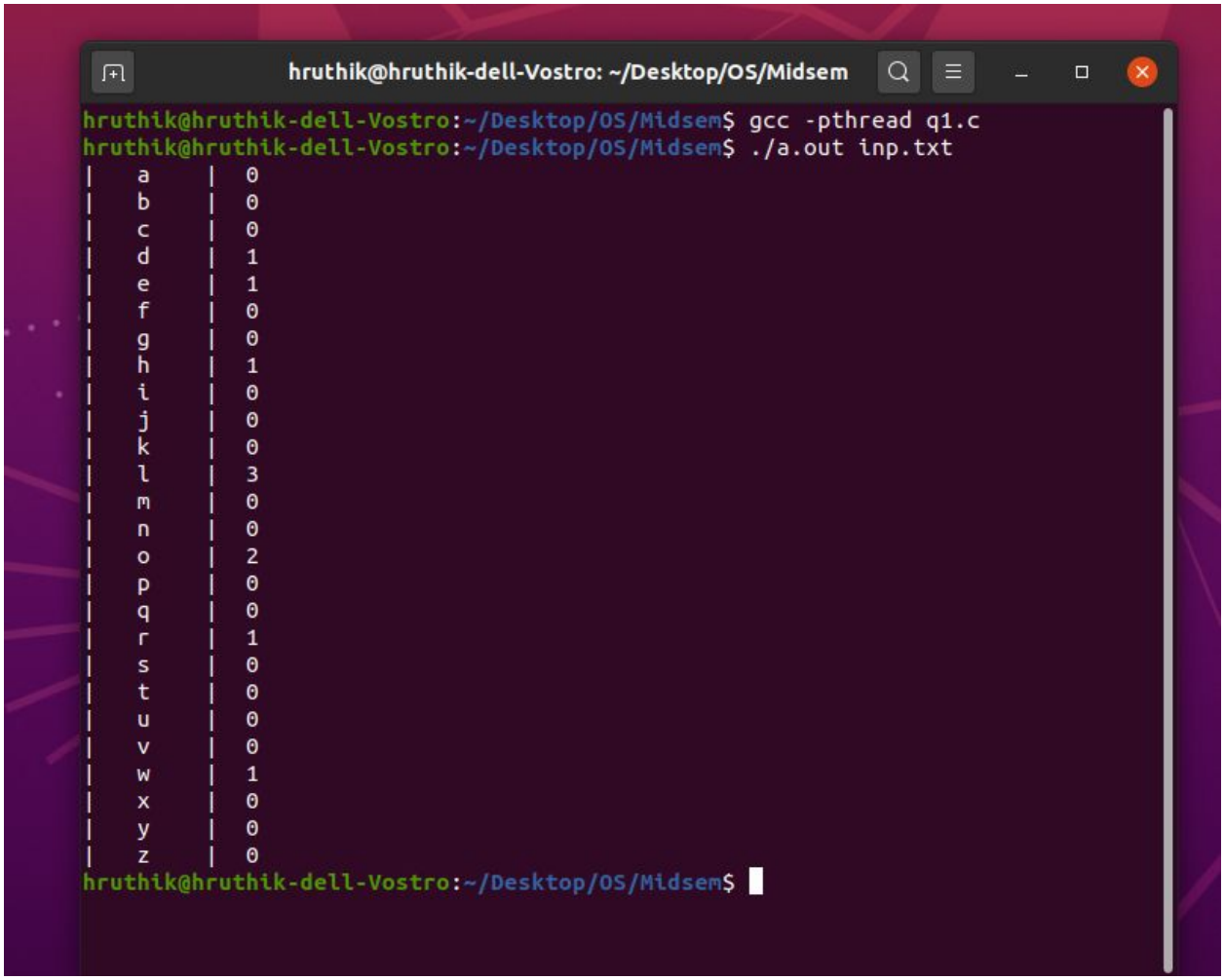
CODE :

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

int c;
int *charCount;
FILE *file;

FILE *openFile(char *filename) // open a file in read mode and return the pointer
```

```
{
        FILE *file;
        file = fopen(filename, "r");

        if (!file)
        {
        printf("Error!\n");
        return NULL;
        }

        return file;
}

void *runner(void *param)
{

        int i = *((int *)param);
        while ((c = tolower(fgetc(file))) != EOF)
        {
        if (i == 26 && (c < 97 || c > 122))
        charCount[c]++; // Count other char
        else if (c == i + 97)
        charCount[i + 97] += 1; // Count letters
        }

        pthread_exit(0);
}

int main(int argc, char *argv[]) //command line arguments
{


        if (argc != 2)
        {
        printf("Syntax: ./a.out <filename>\n");
        return 1;
        }
```

```
        char *filename = argv[1];

        if ((file = openFile(filename)) == NULL)
        return 1;

        pthread_t tid;
        pthread_attr_t attr;

        charCount = mmap(NULL, 128 * sizeof(*charCount), PROT_WRITE,
MAP_SHARED | MAP_ANONYMOUS, -1, 0);
        for (int i = 0; i < 27; i++)
        {

        file = openFile(filename);
        int *arg = malloc(sizeof(*arg));
        *arg = i;
        rewind(file);
        pthread_attr_init(&attr);
        pthread_create(&tid, &attr, runner, arg);
        pthread_join(tid, NULL);
        }
        for (int i = 97; i < 123; i++)
        {
        printf("|   %c        |  %0d ", i, charCount[i]);
        printf("\n");
        }

        if (fclose(file) != 0)
        {
        printf("Error closing file!\n");
        return 1;
        }

        return 0;
}
```

OUTPUT :



THE TEXT USED IN THE CODE IS : “helloworld”

The above code is the multi threading version of Histogram generator to count the occurrence of various characters in a given text.

EFFICIENCY :

The time complexity of the Histogram is :

Time Complexity: **O(n),**
where n is the number of characters in the string.

AMDAHL'S LAW :

In general terms, Amdahl's Law states that in parallelization, if P is the proportion of a system or program that can be made parallel, and 1-P is the proportion that remains serial, then the maximum speedup S(N) that can be achieved using N processors is: S(N)=1/((1-P)+(P/N)) As N grows the speedup tends to 1/(1-P).

The Amdahl's Law calculator computes the speedup of the execution of a task based on the speed up factor (s) of the improvable portion of the task and the proportion (p) of the task that can be improved.

The Math / Science

1. **v = speedup factor.**
2. **P = portion of the task accelerated.**
3. **S = speedup for the portion.**

- **If Pe is the performance for entire task using the enhancement when possible, Pw is the performance for entire task without using the enhancement, Ew is the execution time for entire task without using the enhancement and Ee is the execution time for entire task using the enhancement when possible then,**

  Speedup = Pe/Pw

  Or

Speedup = Ew/Ee

$$\text{Overall Speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

$$= \frac{1}{\left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)}$$

Let's take an example, if the part that can be improved is 30% of the overall system and its performance can be doubled for a system, then −

SpeedupMAX = 1/((1-0.30)+(0.30/2))

= 1.18

Now, in another example, if the part that can be improved is 70% of the overall system and its performance can be doubled for a system, then −

SpeedupMAX= 1/((1-0.70)+(0.70/2))

= 1.54

So, we can see, if 1-p can't be improved, the overall performance of the system cannot be improved so much. So, if 1-p is 1/2, then speed cannot go beyond that, no matter how many processors are used.

- **Multicore programming is most commonly used in signal processing and plant-control systems. In signal processing, one can have a concurrent system that processes multiple frames in parallel. The controller and the plant can execute as two separate tasks, in plant-control systems.**

- **Multicore programming helps to split the system into multiple parallel tasks, which run simultaneously, speeding up the overall execution time.**

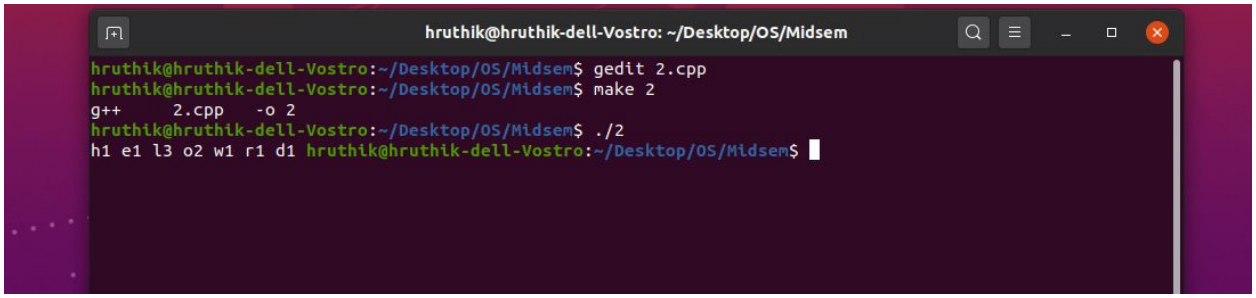
Serial code for Histogram :

```
#include <bits/stdc++.h>

using namespace std;


#define SIZE 26


void printCharWithFreq(string str)

{

        // size of the string 'str'

        int n = str.size();


        // 'freq[]' implemented as hash table

        int freq[SIZE];


        // initialize all elements of freq[] to 0

        memset(freq, 0, sizeof(freq));


        // accumulate freqeuncy of each character in 'str'

        for (int i = 0; i < n; i++)

        freq[str[i] - 'a']++;
```

```
        // traverse 'str' from left to right
        for (int i = 0; i < n; i++) {

        // if frequency of character str[i] is not
        // equal to 0
        if (freq[str[i] - 'a'] != 0) {

        // print the character along with its
        // frequency
        cout << str[i] << freq[str[i] - 'a'] << " ";

        // update frequency of str[i] to 0 so
        // that the same character is not printed
        // again
        freq[str[i] - 'a'] = 0;
        }
        }
}
 int main()
{
        string str = "helloworld";
        printCharWithFreq(str);
```

```
    return 0;
}
```

OUTPUT :



Advantages of Thread

- **Threads minimize the context switching time.**
- **Use of threads provides concurrency within a process.**
- **Efficient communication.**
- **It is more economical to create and context switch threads.**
- **Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.**

NOTE :

Therefore the Multi threading version of histogram is more efficient than the serial version of the histogram.