Jordan University of Science and Technology

Computer science department

Multithreading project

Fadl Masri – 144959

CS375 – Operating Systems

Professor Ahmad Al-Zubi

Student information:

Fadl Fawaz Masri – 144959

Introduction:

This document includes documentation and flowcharts on how multithreading works on matrix manipulation operations such as summation, subtraction, and multiplication.

Furthermore, this document will handle subjects such as process analysis, its speed, and response time using multithreading.

Using multithreading will prove effective in our case of processing large matrices as it will generate much quicker response times and perform tasks efficiently.

Program structure:

This program uses C++ programming language with the support of libraries such as:

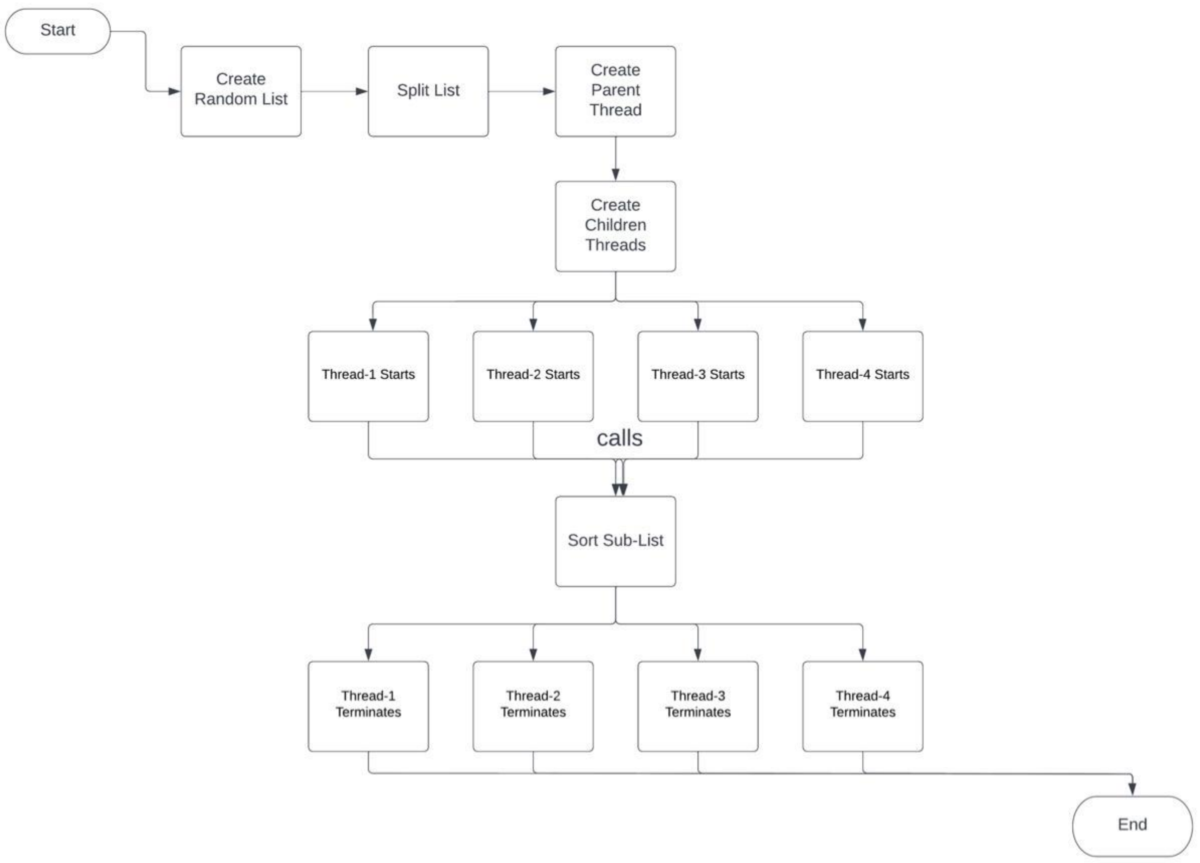
- iostream – used for standard input and output commands.
- pthread – used for creating and joining threads.

Program analysis:

We can clearly observe the great execution time of the program due to using multithreading.

Furthermore, we can see that using normal concurrent processing would result in slow outputs and the output would be exponentially slower as the input size grows.

Program flowchart:



Code package :

```
#include <iostream>
#include <pthread.h>
using namespace std;
using namespace :: chrono;

// amul = Array Multiplication
// asub = Array Subtraction
// ssum = Array Summation

int arr1[10], arr2[10];
int  amul[10], ssum[10], asub[10];
int Mx = 0, Mn = 0;


void *SUM(void *arg)
{
    for (int i = 0; i < 10; i++)
        ssum[i] = arr1[i] + arr2[i];

    cout << "\nthe output of arr1 + arr2 is: ";

    for (int i = 0; i < 10; i++)
        cout << ssum[i] << ' ';

    cout << endl ;

    return NULL;
}

void *SUB(void *arg)
{
    for (int i = 0; i < 10; i++)
        asub[i] = arr1[i] - arr2[i];

    cout << "\nthe output of arr1 - arr2 is: ";

    for (int i = 0; i < 10; i++)
        cout << asub[i] << ' ';

    cout << endl ;

    return NULL;
}

void *MULT(void *arg)
```

```
{
    for (int i = 0; i < 10; i++)
        amul[i] = arr1[i] * arr2[i];

    cout << "\nthe output of arr1 * arr2 is: ";

    for (int i = 0; i < 10; i++)
        cout << amul[i] << ' ';

    cout << endl ;

    return NULL;
}

void *Min(void *arg)
{
    cout << "\nthe minimum between arr1 and arr2 is: ";

    int min1 = arr1[0], min2 = arr2[0];

    for (int ind = 0; ind < 10; ind++)
        if (arr1[ind] < min1)
            min1 = arr1[ind];

    for (int ind = 0; ind < 10; ind++)
        if (arr2[ind] < min2)
            min2 = arr2[ind];

    if (min1 < min2)
        cout << min1 << endl ;
    else
        cout << min2 << endl ;

    return NULL;
}

void *Max(void *arg)
{
    cout << "\nthe maximum between arr1 and arr2 is: ";

    int max1 = arr1[0], max2 = arr2[0];

    for (int ind = 0; ind < 10; ind++)
        if (arr1[ind] > max1)
            max1 = arr1[ind];

    for (int ind = 0; ind < 10; ind++)
        if (arr2[ind] > max2)
        max2 = arr2[ind];

    if (max1 > max2)
        cout << max1 << endl ;
    else
        cout << max2 << endl ;

    return NULL;
}


int main()
{
    pthread_t T1, T2, T3, T4, T5;

    auto start = high_resolution_clock::now(); // time stamp for begining of execution
    cout << " Enter the first array(arr1)" << endl;
    for (int i = 0; i < 10; i++)
        cin >> arr1[i];

    cout << " Enter the second array(arr2)" << endl;
    for (int i = 0; i < 10; i++)
        cin >> arr2[i];

    cout << endl;

    cout << "First array: " << endl;
    for (int y = 0; y < 10; y++)
        cout << arr1[y] << ' ';
    cout << endl;
    cout << "second array: " << endl;
    for (int y = 0; y < 10; y++)
        cout << arr2[y] << ' ';
    cout << endl;

    pthread_create(&T1, NULL, MULT, NULL);
    pthread_join(T1, NULL);
    pthread_create(&T2, NULL, SUM, NULL);
    pthread_join(T2, NULL);
    pthread_create(&T3, NULL, SUB, NULL);
    pthread_join(T3, NULL);
    pthread_create(&T4, NULL, Min, NULL);
    pthread_join(T4, NULL);
    pthread_create(&T5, NULL, Max, NULL);
    pthread_join(T5, NULL);
    auto stop = high_resolution_clock::now();// time stamp for ending of execution
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "\nDuration used to perform all operation is: " << duration.count() << " microseconds." << endl;
    return 0;
}
```