

**A PROJECT REPORT**

**ON**

**Multithreaded Sorting Algorithm**

**for Course IT2230\_OS Lab**

Submitted by

**Aditya Sharma 199302043 IT-4A/2**

**Keshav Yadav 199303068 IT-4A/2**

Supervised By

**Dr. Prakash Chandra Sharma**

**Assistant Professor (Selection Grade)**

**DEPARTMENT OF INFORMATION AND TECHNOLOGY**

**MANIPAL UNIVERSITY JAIPUR**

**JAIPUR-303007 RAJASTHAN, INDIA**

**MAY 2021**

May 2021

# CERTIFICATE

This is to certify that the project titled **Multithreaded Sorting Algorithm** is a record of the bona fide work done by **Aditya Sharma (199302043) & Keshav Yadav (199302068**) submitted for the course **Operating System Lab (IT2230)** in the department of **Information Technology** of Manipal University Jaipur, during the academic session Feb-May 2021.

### Dr. Prakash Chandra Sharma

*Course Instructor & Project Supervisor*

*Dept of Information and Technology Manipal University Jaipur*

### Dr. Pankaj Vyas

*HOD, Dept of Information and Technology*

*Manipal University Jaipur*

**ABSTRACT**

Our project was based on Multithreaded Sorting algorithm. Threads are popular way to improve application through parallelism. Threads operate faster than processes due to following reasons:

1) Thread creation is much faster.

2) Context switching between threads is much faster.

3) Threads can be terminated easily

4) Communication between threads is faster.

Merge Sort is a popular sorting technique which divides an array or list into two halves and then start merging them when sufficient depth is reached. Time complexity of merge sort is O (n logn).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Contents** | | | | |
|  | | | | Page No |
| **Chapter 1** | | | **INTRODUCTION** |  |
|  | **1.1** | Introduction to Work Done | |  |
|  | **1.2** | Project Statement | |  |
| **Chapter 2** | | | **BACKGROUND DETAIL** |  |
|  | **2.1** | Conceptual Overview | |  |
|  | **2.2** | Hardware & Software Requirement | |  |
| **Chapter 3** | | | **METHODOLOGY** |  |
|  | **3.1** | Flow Chart/ Block Diagram | |  |
|  | **3.2** | Methodology/Algorithm/Procedure | |  |
| **Chapter 4** | | | **IMPLEMENTATION** |  |
|  | **4.1** | Implementation Modules | |  |
|  | **4.2** | Coding | |  |
| **Chapter 5** | | | **RESULTS AND ANALYSIS** |  |
| **REFERENCES** | | | |  |

**1.1 Introduction to Work Done**

This is a simple C program to sort a given array using multithreading. We have used two threads to sort arrays individually and then finally merging them together to get a final output, this can also be considered as a variation of Merge Sort.

This would be done by firstly taking input from user and then, passing these values into our function through a globally stored array which will hold the values given by user.

These values will be equally divided and sorted by a thread each, and finally merged.

**1.2 Project Statement**

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term sorting threads) sort each sub list using a sorting algorithm of your choice. The two sub lists are then merged by a third thread—a merging thread—which merges the two sub lists into a single sorted list. Because global data are shared cross all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sub lists into this second array.

**2. BACKGROUND DETAIL**

**2.1** **Conceptual Overview**

Multithreading is a phenomenon of executing multiple threads at the same time. To understand the concept of multithreading, you must understand what is a thread and a process.

Creating a thread is less expensive than creating a new process because the newly created thread uses the current process address space. The time that is required to switch between threads is less than the time required to switch between processes. A switch between threads is faster because no switching between address spaces occurs. Communication between the threads of one process is simple because the threads share everything, most importantly address space. So, data produced by one thread is immediately available to all the other threads in the process.

However, this sharing of data leads to a different set of challenges for the programmer. Care must be taken to synchronize threads to protect data from being modified by more than one thread at once, or from being read by some threads while being modified by another thread at the same time. See Thread Synchronization for more information.

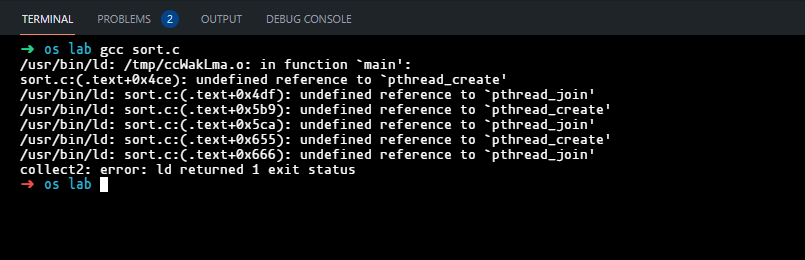
**2.2 Hardware & Software Requirement**

No special requirements needed for the hardware part considering today’s standards for computers and laptops or machines in general.

On the software front, user is required to have gcc compiler downloaded and set to path

for execution of .c files.

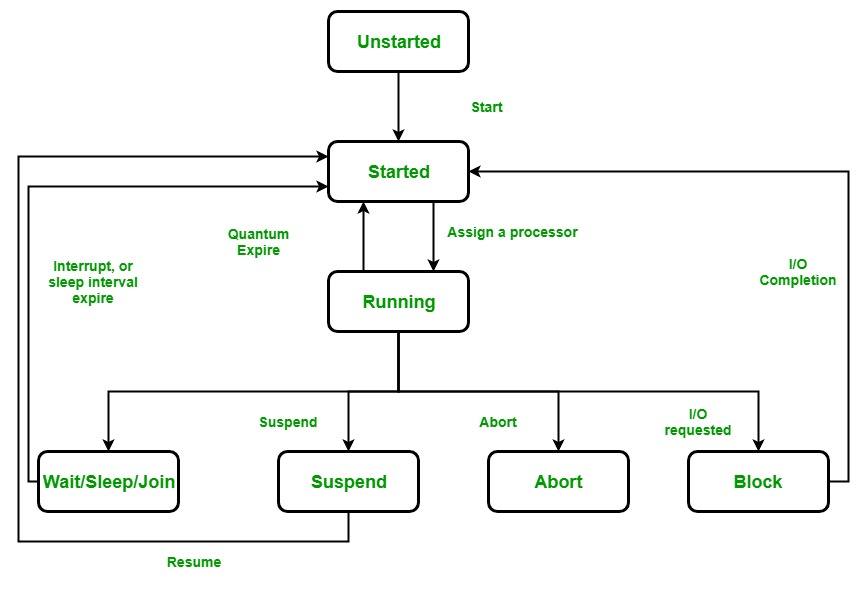
While executing the program, user might face errors of the sort given below.

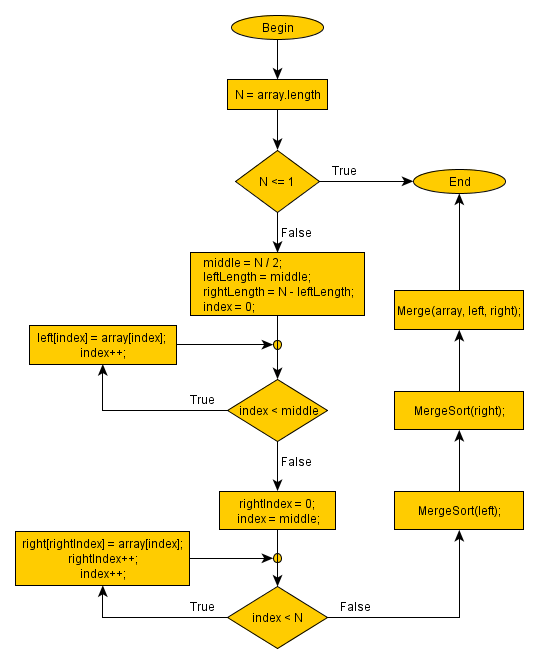


To avoid this error, user is expected to add “-pthread” to the end of their command on the command line while compiling the program. This is because -pthread is not natively supported in Windows machines but this could be countered by adding -pthread to execute the program.

**3. METHODOLOGY**

**3.1 Flow Chart/ Block Diagram**





**3.2** **Methodology/Algorithm/Procedure**

**4. IMPLEMENTATION**

**4.1** **Implementation Modules**

* stdio.h
* pthread.h
* stdlib.h
* string.h

**4.2 Coding**

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <string.h>

int array\_ptr[10];

int b[10];

int c[5], d[5];

int low = 0, high = 9, high1 = 4;

void \*run(void \*array\_ptr);

void \*run1(void \*array\_ptr);

void merge1(int low, int mid, int high1)

{

    int l1, l2, i;

    for (l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high1; i++)

    {

        if (c[l1] <= c[l2])

            d[i] = c[l1++];

        else

            d[i] = c[l2++];

    }

    while (l1 <= mid)

        d[i++] = c[l1++];

    while (l2 <= high1)

        d[i++] = c[l2++];

    for (i = low; i <= high1; i++)

    {

        c[i] = d[i];

    }

}

void mergesort1(int low, int high1)

{

    int mid;

    int \*elements = (int \*)c;

    if (low < high1)

    {

        mid = (low + high1) / 2;

        mergesort1(low, mid);

        mergesort1(mid + 1, high1);

        merge1(low, mid, high1);

    }

    else

    {

        return;

    }

}

int c1[5], d1[5];

void \*merging(void \*array\_ptr)

{

    int i = 0;

    int j = 0;

    int k = 0;

    int m = 5, n = 5;

    while (i < m && j < n)

    {

        if (c1[i] < d1[j])

        {

            b[k] = c1[i];

            i++;

        }

        else

        {

            b[k] = d1[j];

            j++;

        }

        k++;

    }

    if (i >= m)

    {

        while (j < n)

        {

            b[k] = d1[j];

            j++;

            k++;

        }

    }

    if (j >= n)

    {

        while (i < m)

        {

            b[k] = c1[i];

            i++;

            k++;

        }

    }

    printf("\n After merging: \n");

    for (i = 0; i < m + n; i++)

    {

        printf("\n%d", b[i]);

    }

}

int main()

{

    int n = 10;

    printf(" 10 elements you want to enter\n");

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &array\_ptr[i]);

    }

    for (int i = 0; i < 5; i++)

    {

        c[i] = array\_ptr[i];

    }

    printf("\n 2 sorted subarray are:\n ");

    pthread\_t t1, t2, t3;

    pthread\_create(&t1, NULL, run, (void \*)c);

    pthread\_join(t1, NULL);

    for (int i = 0; i < 5; i++)

    {

        printf("%d\t", c[i]);

        c1[i] = c[i];

    }

    printf("\n");

    for (int i = 0; i < 5; i++)

    {

        c[i] = array\_ptr[i + 5];

    }

    pthread\_create(&t2, NULL, run, (void \*)c);

    pthread\_join(t2, NULL);

    for (int i = 0; i < 5; i++)

    {

        printf("%d\t", c[i]);

        d1[i] = c[i];

    }

    pthread\_create(&t3, NULL, merging, (void \*)array\_ptr);

    pthread\_join(t3, NULL);

}

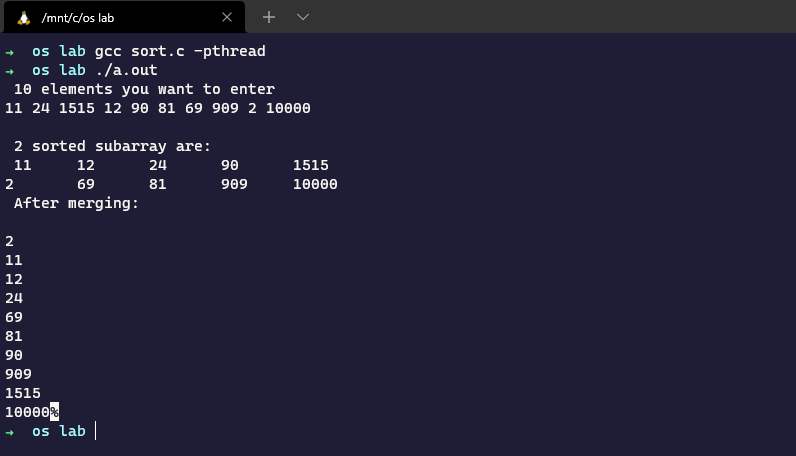
void \*run(void \*c)

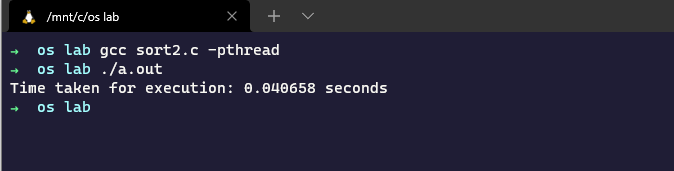
{

    mergesort1(low, high1);

}

**5. RESULTS AND ANALYSIS**

****

****

While making this project, we found out about multithreading and how it can be used to make a program more efficient. Also, we found out about Merge sort algorithm and how it can be used to sort an array of numbers.

Using multiple threads, one can divide the work load of a given program and efficiently reduce time and space complexity and can be used for extremely large size of arrays to sort them quickly.