

Group Project 3: Application for Threads Sorting

Antonio Hughes (015748783)

Sandra Chavez (016363540)

CECS 326 - Sec 04

Prof. Hailu Xu

11-8- 2020

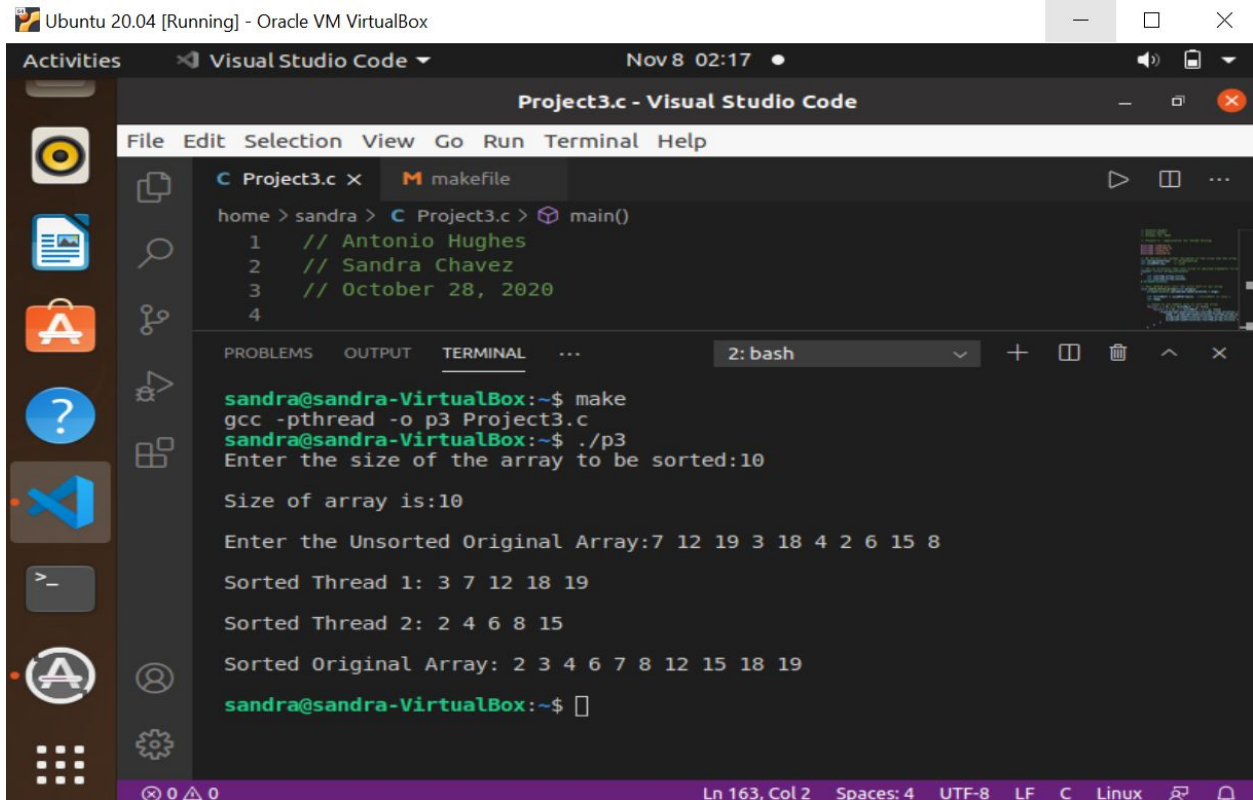
Abstract

This project focuses on the implementation of threads and on designing an application with multiple threads. Similar to the previous projects, we are expected to use the threads programming interface: POSIX Threads (Pthreads). This project asks us to write a multithreaded sorting program. A list of integers will be divided into two halves and two threads will sort each of the two sublists. The two sorted sublists will then be merged into a single sorted list by a third thread. Data was shared between the threads by using global arrays. The implementation on how we are to sort our threads was left up to us. We decided to split the array into two halves and sort it using bubble sort and then merging the two arrays again sorting it into its final result. This allowed us to be able to sort any thread, regardless of the size of the array.

Purpose

The purpose of this lab assignment was to demonstrate the topic of thread sorting. With an array split into two halves, each thread would be in charge of sorting each half and then merging them into one array. This project allows us to observe the results of how threads can be sorted simultaneously to save on run time for calculations versus sorting a thread one at a time. Using multiple threads increases parallelism by running the threads concurrently in different cores of the processor. Although the differences of the results may seem small to us in run time, this would actually make a huge difference in the efficiency of all runtimes in the operating system. It would allow more to be done efficiently without any risk or other conflicting issues.

Results



```
home > sandra > C Project3.c > main()
1 // Antonio Hughes
2 // Sandra Chavez
3 // October 28, 2020
4

PROBLEMS OUTPUT TERMINAL ... 2: bash

sandra@sandra-VirtualBox:~$ make
gcc -pthread -o p3 Project3.c
sandra@sandra-VirtualBox:~$ ./p3
Enter the size of the array to be sorted:10

Size of array is:10

Enter the Unsorted Original Array:7 12 19 3 18 4 2 6 15 8

Sorted Thread 1: 3 7 12 18 19
Sorted Thread 2: 2 4 6 8 15
Sorted Original Array: 2 3 4 6 7 8 12 15 18 19
sandra@sandra-VirtualBox:~$
```

This screenshot above demonstrates the program asking the user for a specific size of the array, the elements itself and then proceeds to sort the array.

Contributions by group members:

- **Antonio Hughes**
 - Made the MakeFile
 - Worked on the report (purpose / analysis)
 - Worked on the main (user input, creating/joining the threads)
- **Sandra Chavez**
 - Wrote the ReadMe file
 - Worked on the report (abstract / results)
 - Worked on the sorting function (utilized a struct to contain the two subarrays and sorted the two subarrays)

Analysis

Our results of our lab assignment were on track with the expected output from the prompt. Based on our design of the code, we were able to take in any number N as the size of the array. With that we reassure we have the correct size by displaying it, for better testing purposes. With that, we are then able to enter the unsorted array to be sorted. Using the example given on the prompt, we were successfully able to split the original array up into two halves, sort accordingly, and then merge it all back together properly displaying the correct output. With this we can observe the usefulness of threads being sorted. With this ability, we can see how sorting threads in general achieves parallelism. Sorting multiple threads would allow us to achieve more by using multiple cores and serving more clients and serving them faster than sorting singular threads.