

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | 冯春霖 |
| **Student ID:** | 222019321062074 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Pthread Library and Concurrent Programming | | | |
| Date | | April，2021 | Type | | √ Confirmatory  √ Design  √ Comprehensive |
| 1. **Objective & Requirements**    1. Grasp the Pthreads API for thread creation, termination operations    2. Grasp concurrent programming skills | | | | | |
| 1. **Experimental environment (**platform and software**)**   Virtualbox + Ubuntu (or other platform+linux system combinations) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results) 2. Task1    * 1. Create a new thread in the main thread      2. Pass to integers to the new thread and calculate the sum of the two integers by the new thread 3. Task2    * 1. Define an integer array of length 200000      2. Randomly initialize the integer array      3. Sort the initialized integer array and measure the time cost (hint: use the time command) 4. Task3    * 1. Write a C program to merge two sorted integer arrays in to a single sorted integer array 5. Task4   Write a multithreaded sorting program that works as follows:   * + 1. Set the number of CPUs of your virtual machine to at least 2 in VirtualBox     2. Define two GLOBAL integer arrays **a** and **b**, both of length 200000     3. Randomly initialize the array **a**     4. In the main thread, create two new threads to sort the first half and the second half of array **a** respectively     5. The main thread waits for the two new threads to terminate, and then merge the sorted first and second half of array **a** into array **b**     6. Compare the time cost of your multithreaded program with the time you obtained in Task 2, and compute the speedup. (hint: **time** command)      1. Please provide your procedure and source codes to perform the tasks.   **Task1:**  The code is shown in the screenshot, we can see that the parameters are passed into:    The result of the program is shown in the screenshot, he parameters are passed in and can be calculated correctly    **Task2:**  Initialize the array by generating a random number using the init() function:    I used a quick sort algorithm to accomplish this task, with the following code:    When I looked at the time consumed by the program, I found that the quick sort was so efficient that the run time was too short for subsequent experiments, so I changed the SIZE to 2000000:    **Task3:**  I wrote the code for the merge() function as shown in the screenshot:    The result of the merge() function is shown in the screenshot. I use p1 and p2 to represent the two arrays before merging and res to represent the merged array, and output them separately in the main function. We can see that both sorting and merging are successful.    **Task4:**  The main function of the program is shown in the screenshot, creating two threads to call runner(), then calling the sorting algorithm via runner(), and finally waiting for the threads to return and merge in the main function. The sorting and merging functions follow the same content as in Task2 and Task3.    The result of running the program is shown in the screenshot, where the size of the data is SIZE=2000000, the former one is the result of single-threaded sorting, and the latter one is the result of merging after multi-threaded sorting. **speedup** = 0.258/0.159 = **1.623**. | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Through this experiment, I got to know about the thread library pthread and learned to use this library in a simple way. At the same time I gained an understanding of concurrent programming using multiple threads. Not only that, but I also gained a deeper understanding of the time complexity that I had previously learned in the data structures course, in Task2, it took more than two minutes for 2E5 data scale for students using bubble sort, but less than three seconds for 2E6 data scale for me using quick sort. However, there is still a question about this experiment, for larger data scale (more than 1E8), using a single thread to sort can be completed normally, and the time is not too long, but when running multi-threaded will get an error, I still have not found the exact answer to this, I guess it may be due to the stackoverflow. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |