Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2019

Homework 7 – Summation using Threads

Due: 10/05/2019, Friday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to write a **multithreaded** C++ program that reads numbers from a file and adds them up concurrently using threads. The program starts after taking some inputs from the keyboard and continues until all the numbers are added. In your program, you are going to display some verbose output about addition operations and threads. Please see "Details of the Program" section for details.

Using Threads

There will be **five threads** (other than main thread) in your program. **All of them** will wait their turn and add the next value. You must ensure that at a given time only one thread will make an addition operation to prevent race conditions and obtain a correct result in the end. For this, you are required to use *mutexes*. In the end, your threads must be joined properly, and your program must terminate without any complications.

Details of the Program

Before the summation begins, three inputs are entered via keyboard. First, the file name which contains the numbers to be added, is entered. Then, the minimum and maximum waiting time before each addition operation for a thread are entered in seconds (two inputs).

After the inputs are entered, your program will read the numbers from the given file name and store them in a dynamic array. Each line in the file contains only one number. The first line contains the amount of numbers in the file; you can use this information, which is the size of the array, when creating your dynamic array. The remaining lines contain numbers to be added.

You may assume that all inputs are correct, including the file structure, so no input checks are necessary. That means you can assume that the entered file name exists, it has a correct structure, the min and max waiting times are entered as positive integer values and the first one is less than or equal to the second one.

In the next step, you can create and start the five threads. The entry point of each of the threads will be the same function. That means, you will **NOT** write separate functions for the threads.

At the beginning of the summation, display a message that says that the summation has started and the time of start (see "Sample Runs" for examples).

For each thread, before each addition operation, the thread must wait a random amount of time (in seconds). Then a thread will try to enter the critical area marked by a mutex. If it succeeds to get the lock of the mutex, it will make the next addition operation and store its result in the shared sum. Then it will increment the shared index count and will exit the critical region. The total number of addition operation that a particular thread will make is not fixed; it depends on the total number of elements to be added, random waiting times of the threads and the scheduling of the threads. You have to implement the entry point function for the threads (remember, there will be only one thread function) in such a way that in the critical region the next value will be added to the sum. To do so, you have to keep a shared sum, as well as a shared index counter to track down the element to be added next. These shared sum and index count could be made as global.

The random waiting times before the addition operation are determined via a function that returns a random integer between (and including) a minimum and maximum values passed as parameters to the thread function from main. **Do not use the `RandGen` class from** CS201 for random integer generation in multithreaded applications because it is not thread-safe. Instead, **you must use the following thread-safe function, `random_range`, for generating random waiting times inside the threads**. This function returns a random number between (and including) `min` and `max` parameters. In the threads, you can call it by passing minimum and maximum waiting times of the corresponding thread as arguments.

```
#include <random>
#include <time.h>

int random_range(const int & min, const int & max) {
    static mt19937 generator(time(0));
    uniform_int_distribution<int> distribution(min, max);
    return distribution(generator);
}
```

To pause a thread for a certain amount of time, you should use the `this_thread::sleep_for(chrono::seconds(`*time_in_seconds*`))` command; you need to include the `thread` and `chrono` libraries in your program for this to work. In order to get the current time and to display it, we have seen some codes in class; you may use them.

The thread function and consequently the program will end after all of the numbers are added and the summation result is obtained. At the end of your program, do not forget to join five threads properly. Moreover, display a message to specify that the summation has ended including the result and the ending time.

Use of global variables

You may use global variables in this homework. Actually, it would be miserable not to use them in a program that has several threads. However, we kindly request you not to exaggerate the global usage since after a certain point you may lose control over your program (as the famous Turkish proverb says "azı karar, coğu zarar").

Sample Runs

Some sample runs are given below, but these are not comprehensive, therefore you have to consider all cases, to get full mark.

Due to the probabilistic nature of the homework and due to the scheduling of threads, same inputs may yield different outputs for your code. However, the order of the events must be consistent with the homework requirements and the given inputs.

Sample input files are given together with homework document in the zip file.

The inputs from the keyboard are written in ***boldface and italic***.

Sample Run 1:

```
Please enter the file name.
input1.txt
Please enter the wait range of threads.
1 3

Starting reading the array at 17:54:52
Array stored in the memory. Starting the summation at 17:54:52

Thread 2 added number index 0 to the total sum at 17:54:53
Current sum: 3
Thread 5 added number index 1 to the total sum at 17:54:53
Current sum: 8
Thread 1 added number index 2 to the total sum at 17:54:54
Current sum: 14
Thread 3 added number index 3 to the total sum at 17:54:54
Current sum: 18
Thread 2 added number index 4 to the total sum at 17:54:54
Current sum: 21
Thread 4 added number index 5 to the total sum at 17:54:55
Current sum: 29
Thread 1 added number index 6 to the total sum at 17:54:55
Current sum: 38
Thread 5 added number index 7 to the total sum at 17:54:56
Current sum: 137
Thread 1 added number index 8 to the total sum at 17:54:56
Current sum: 143
Thread 3 added number index 9 to the total sum at 17:54:57
Current sum: 150

Adding finished at 17:54:59
Sum: 150
```

Sample Run 2:

```
Please enter the file name.
input1.txt
Please enter the wait range of threads.
2 2
```

```
Starting reading the array at 18:02:31
Array stored in the memory. Starting the summation at 18:02:31

Thread 1 added number index 0 to the total sum at 18:02:33
Current sum: 3
Thread 2 added number index 1 to the total sum at 18:02:33
Current sum: 8
Thread 3 added number index 2 to the total sum at 18:02:33
Current sum: 14
Thread 4 added number index 3 to the total sum at 18:02:33
Current sum: 18
Thread 5 added number index 4 to the total sum at 18:02:33
Current sum: 21
Thread 1 added number index 5 to the total sum at 18:02:35
Current sum: 29
Thread 2 added number index 6 to the total sum at 18:02:35
Current sum: 38
Thread 3 added number index 7 to the total sum at 18:02:35
Current sum: 137
Thread 4 added number index 8 to the total sum at 18:02:35
Current sum: 143
Thread 5 added number index 9 to the total sum at 18:02:35
Current sum: 150

Adding finished at 18:02:37
Sum: 150
```

Sample Run 3:

```
Please enter the file name.
input1.txt
Please enter the wait range of threads.
3 8

Starting reading the array at 18:00:55
Array stored in the memory. Starting the summation at 18:00:55

Thread 5 added number index 0 to the total sum at 18:01:00
Current sum: 3
Thread 1 added number index 1 to the total sum at 18:01:01
Current sum: 8
Thread 3 added number index 2 to the total sum at 18:01:01
Current sum: 14
Thread 2 added number index 3 to the total sum at 18:01:02
Current sum: 18
Thread 4 added number index 4 to the total sum at 18:01:03
Current sum: 21
Thread 5 added number index 5 to the total sum at 18:01:03
Current sum: 29
Thread 2 added number index 6 to the total sum at 18:01:06
Current sum: 38
Thread 4 added number index 7 to the total sum at 18:01:06
Current sum: 137
Thread 5 added number index 8 to the total sum at 18:01:06
```

```
Current sum: 143
Thread 1 added number index 9 to the total sum at 18:01:08
Current sum: 150

Adding finished at 18:01:14
Sum: 150
```

Sample Run 4:

```
Please enter the file name.
input2.txt
Please enter the wait range of threads.
1 9

Starting reading the array at 18:05:59
Array stored in the memory. Starting the summation at 18:05:59

Thread 2 added number index 0 to the total sum at 18:06:00
Current sum: -4
Thread 3 added number index 1 to the total sum at 18:06:01
Current sum: -11
Thread 4 added number index 2 to the total sum at 18:06:02
Current sum: -12
Thread 5 added number index 3 to the total sum at 18:06:06
Current sum: -2
Thread 2 added number index 4 to the total sum at 18:06:06
Current sum: -10
Thread 3 added number index 5 to the total sum at 18:06:06
Current sum: -12
Thread 5 added number index 6 to the total sum at 18:06:07
Current sum: -7

Adding finished at 18:06:13
Sum: -7
```

Sample Run 5:

```
Please enter the file name.
input2.txt
Please enter the wait range of threads.
1 1

Starting reading the array at 18:07:38
Array stored in the memory. Starting the summation at 18:07:38

Thread 1 added number index 0 to the total sum at 18:07:39
Current sum: -4
Thread 2 added number index 1 to the total sum at 18:07:39
Current sum: -11
Thread 3 added number index 2 to the total sum at 18:07:39
Current sum: -12
Thread 4 added number index 3 to the total sum at 18:07:39
Current sum: -2
Thread 5 added number index 4 to the total sum at 18:07:39
```

```
Current sum: -10
Thread 1 added number index 5 to the total sum at 18:07:40
Current sum: -12
Thread 2 added number index 6 to the total sum at 18:07:40
Current sum: -7

Adding finished at 18:07:41
Sum: -7
```

Sample Run 6:

```
Please enter the file name.
input2.txt
Please enter the wait range of threads.
1 2

Starting reading the array at 18:04:59
Array stored in the memory. Starting the summation at 18:04:59

Thread 2 added number index 0 to the total sum at 18:05:00
Current sum: -4
Thread 4 added number index 1 to the total sum at 18:05:00
Current sum: -11
Thread 1 added number index 2 to the total sum at 18:05:01
Current sum: -12
Thread 3 added number index 3 to the total sum at 18:05:01
Current sum: -2
Thread 5 added number index 4 to the total sum at 18:05:01
Current sum: -10
Thread 2 added number index 5 to the total sum at 18:05:01
Current sum: -12
Thread 1 added number index 6 to the total sum at 18:05:02
Current sum: -7

Adding finished at 18:05:03
Sum: -7
```

Please see the previous homework specifications for the other important rules and the submission guidelines

Last, but not the least, you have to use Windows for this homework. In multithreaded applications, there are several dependencies to the underlying operating system. Thus the behavior that you see while running in a Windows computer could be totally different than MacOS. For this reason, you have to use a Windows computer for this homework. In the previous homework assignments, we kind of tolerated Mac vs. Windows compatibility issues after grading; but in this homework we will not do so. Your code will be tested in Windows and if it does not work, even if it works using Mac, you will not be able to object.

Good Luck!
Albert Levi, Taha Atahan Akyıldız