Sabancı University
Faculty of Engineering and Natural Sciences

CS204 Advanced Programming
Fall 2016-2017
Homework 8 – Multithreading
Due: 25/12/2016 - 23:00
(One day late submission penalty: -10%)

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 HAVE TO write down the code on your own.
You CANNOT HELP any friend while coding.
Plagiarism will not be tolerated!

1. Introduction

The aim of this homework is to get you familiar with multithreading with C++. You will implement a multithreaded program for a bitcoin mining simulation. Bitcoin is a digital currency which has gained popularity in the last five years. To mine Bitcoins, miners try to solve hard mathematical problems by trying random solutions and the first one who finds the solution earns some bitcoins. Additional information can be found at https://www.bitcoinmining.com/

In this homework, miners will be threads. They will compete to update a transaction/operation list. You can consider the list as a chain of transactions. To add a new transaction, the new transaction ID and some information on the last added transaction need to be taken into account. With these inputs, each thread will try to solve a hard mathematical function using a brute-force technique. In other words, each thread will randomly try possible answers to find solutions. The first one to find the answer adds the transaction and gains a bitcoin. Since the problem is updated, i.e., the last transaction in the chain is changed, the other miners will be alerted and all will try to solve a different problem. At the end of the program, the number of bitcoins for each miner will be printed out.

2. Bitcoin Mining

In this homework, each thread will be a miner as mentioned. Miners will use a *Hash* function to calculate a hard problem. Hash functions are known to be one way functions such that given a hash function **f**(x), its computationally feasible to compute **f**(x) but it's infeasible to find $f^{-1}(x)$. For more information on hash functions, read https://en.wikipedia.org/wiki/Hash_function). The hash function (given in the cpp file) takes a transaction and calculates the hash value for that transaction. If this hash value is less than a threshold, i.e., its first n bits are 0, it will be accepted as a solution. The hash function is given and you don't need to change it. If a miner finds a desired solution, it gains a bitcoin. Moreover, as mentioned above, the winner miner will append the corresponding transaction to the chain of transactions, which will be in a linked-list form.

3. Program Flow

At the beginning, user will be prompted 3 inputs:

- Difficulty of mining: between 1-10 (this will decide how hard is the mining problem).
- File name from which the transaction IDs are going to be read
- Number of miners (number of threads created)

After taking these inputs you will first calculate the threshold. Threshold will be an unsigned integer for which first n left-most bits are 0 where n is the difficulty. For example, if the difficulty is 4, the threshold will be 0000 1000 0000 0000 0000 0000 0000 0000. Hence if the difficulty increases, it will be harder to find a smaller number less than the threshold.

Then you will read inputs, the transaction IDs, from the given file. The first line of the file will be the number of entries in the file so you don't have to count how many lines there are. These inputs will be put into an array of **unsigned int** which will be shared among threads by the main thread (i.e., no multithreading until this point).

Finally, your program will create miner threads of desired amount. After creating all the threads, main thread should set a flag to let miners start. Note that, all the threads should wait for this flag to be set, to start mining. You can use **this_thread::yield()** function to stall a thread.

4. Data Structures

In this homework miners will have *transactions* and accepted transactions are stored in a *chain of transactions* which is basically a linked list of transactions. Transactions will have 4 elements in them:

- The number, transaction ID, read from the file
- The *hash* of previous transaction node
- A random number (solution, the number being tried by that thread)
- The thread id inserted the transaction to the chain

You should put all this information in a transaction **struct**. Moreover, you should implement a linked list of transactions to represent the *chain of transactions.* The first transaction in the list will be added manually in the main thread. It should have 0's for the first three struct members, and the id of main thread as thread id.

5. Miner Threads

In this homework, you will implement a *mine* function that will take the following as input:

- The chain of transactions
- The array of input set (read from the file)
- Threshold
- A <u>shared</u> **mutex** object
- A <u>shared</u> integer to keep track of which transaction is added last
- A boolean flag

After the flag is set to true, all of the threads will start trying random values to find a solution. Threads will create transactions using:

- Current item from input set
- Hash of previous transaction
- A random number (You should use **rand()**)
- Thread id

This transaction will be hashed using the hash function. If the hash is less than the threshold, corresponding thread should add that transaction to the chain. The thread finding the solution should increment its **private** bitcoin count by one. Also, it should increment the **shared** integer to notify other threads that a solution is found. Note that multiple threads can find solutions at the same time and only **one** of them should add the transaction to the chain. (Hint: use mutex) If one thread finds a solution but another thread has already added a transaction to the chain and *updated the shared integer*, then the first thread should not add a transaction for the same entry and skip to the next one. When there are no more entries left, each thread should report their bitcoin count one by one. (Hint: again use mutex)

The mine function will be executed by every thread. All the miners (threads) will attack the same entry in the input set. When a solution is found, corresponding thread will update the list and every thread will skip to the next entry. This can be done by keeping the track of a shared integer, indicating which entry should be attacked. The thread finding the solution can update this value and other miners can skip to the next entry.

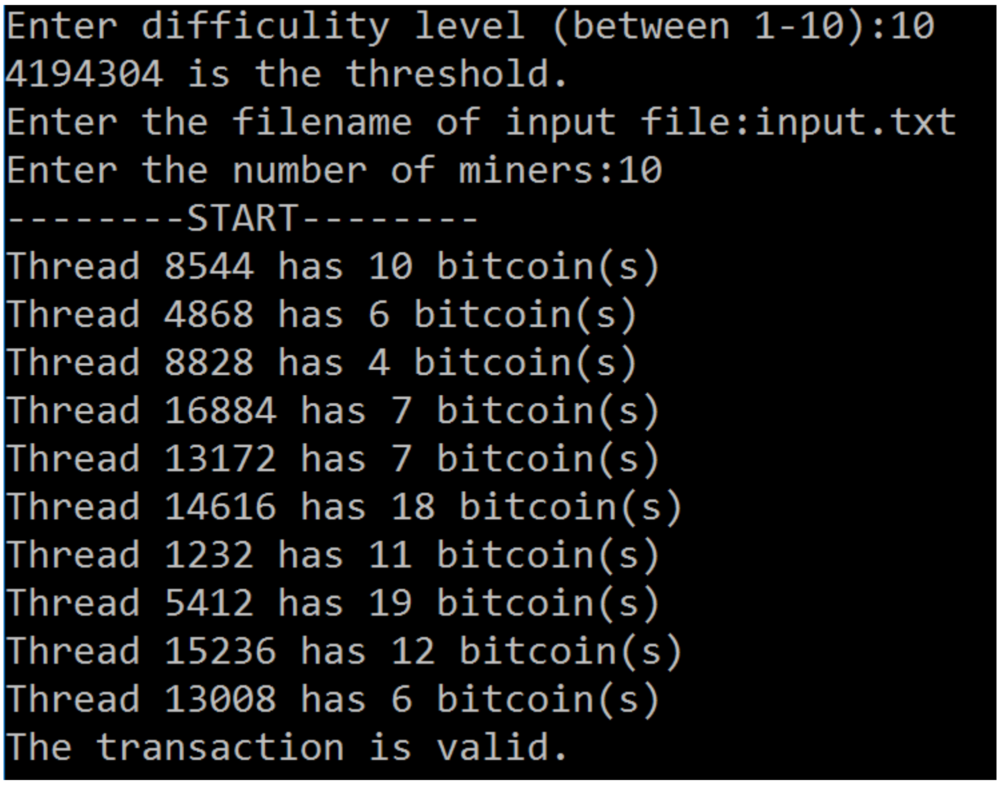
6. Evaluation

You will be provided with a **transactionValidator()** function that takes a transactionChain and a threshold and checks if the transaction chain is valid. Since a transaction node is added by using the hash of previous transaction, the chain is really a chain. The adjacent nodes are strongly tied (the strength depends on the hardness of the problem) and you cannot insert/delete/change a single node without disrupting the validity. You can check the correctness of your program by using this function. You should also print an appropriate message stating that the transaction is valid or not at the end of your program.

7. Sample Output

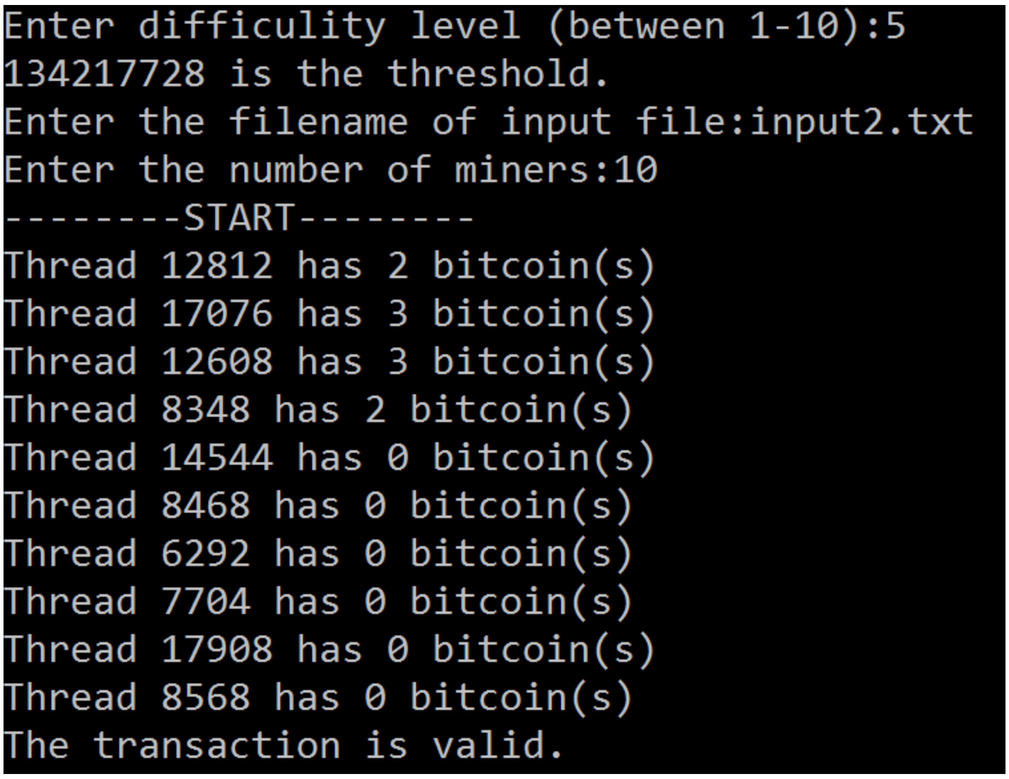
Below, some sample outputs are provided. Since we are using multiple threads and there is a randomness while mining bitcoins, your output might not be the same as given samples. But the format should be the same.

Sample 1: A scenario with 100 entries.



```
Enter difficulity level (between 1-10):10
4194304 is the threshold.
Enter the filename of input file:input.txt
Enter the number of miners:10
--------START--------
Thread 8544 has 10 bitcoin(s)
Thread 4868 has 6 bitcoin(s)
Thread 8828 has 4 bitcoin(s)
Thread 16884 has 7 bitcoin(s)
Thread 13172 has 7 bitcoin(s)
Thread 14616 has 18 bitcoin(s)
Thread 1232 has 11 bitcoin(s)
Thread 5412 has 19 bitcoin(s)
Thread 15236 has 12 bitcoin(s)
Thread 13008 has 6 bitcoin(s)
The transaction is valid.
```

Sample 2: A scenario with 10 entries.



```
Enter difficulity level (between 1-10):5
134217728 is the threshold.
Enter the filename of input file:input2.txt
Enter the number of miners:10
--------START--------
Thread 12812 has 2 bitcoin(s)
Thread 17076 has 3 bitcoin(s)
Thread 12608 has 3 bitcoin(s)
Thread 8348 has 2 bitcoin(s)
Thread 14544 has 0 bitcoin(s)
Thread 8468 has 0 bitcoin(s)
Thread 6292 has 0 bitcoin(s)
Thread 7704 has 0 bitcoin(s)
Thread 17908 has 0 bitcoin(s)
Thread 8568 has 0 bitcoin(s)
The transaction is valid.
```

Some Important Rules:

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate.

When we grade your homeworks we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we may run your programs in *Release* mode and **we may test your programs with very large test cases**.

What and where to submit (PLEASE READ, IMPORTANT): You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.
Name your cpp file that contains your program as follows:

"SUCourseUserName_YourLastname_YourName_HWnumber.cpp"

Your SUCourse user name is actually your SUNet username that is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

Cago_Ozbugsizkodyazaroglu_Caglayan_hw2.cpp

Do not add any other character or phrase to the file name. Make sure that this file is the latest version of your homework program. Compress this cpp file using WINZIP or WINRAR programs. Please use "zip" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains your cpp file.

You will receive no credits if your compressed zip file does not expand or it does not contain the correct file. The naming convention of the zip file is the same as the cpp file (except the extension of the file of course). The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourName_HWnumber.zip

For example zubzipler_Zipleroglu_Zubeyir_hw1.zip is a valid name, but

hw1_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
CS204 Team (Mustafa Kemal Taş, Kamer Kaya)