CS342 - Spring 2019
Project #2
Multi-threaded Programs and Synchronization

Assigned: **March 14, 2019.**
Due date: **March 30, 2019, 23:55.**

Objectives

- Multithreaded programming with Pthreads (POSIX threads).
- Practicing synchronization, use of lock variables.
- Designing and performing experiments; applying probability and statistics knowledge.

You can do this project in groups of two students each. You will use C and Linux.

Thread-safe Hash Table Library

In this project you will first develop a library (`libhash.a`) that will implement *thread-safe hash table* data structure and its operations. A multi-threaded application using the library will be able to create one or more hash tables and access them from multiple threads running concurrently. The library code will sit in a file called `hash.c` and the respective header file (i.e., library interface) will be `hash.h`.

A hash table has N buckets (buckets 0 through N-1), i.e., N entries. A key i and an associate value (data), i.e., a key-value pair, will be inserted into bucket j = hash(i), where j is in range [0, N-1]. In this project, the hash function will be a simple hash function, i.e., hash (i) = i mod N. The key type is *integer* and valid keys are positive. A key value 0 is not valid. The value type will be `void *`. With proper type casting, you can store an integer value in a field that is of type void *. Multiple key-value pairs mapping to the same bucket will be added to a linked list (chaining). In this way collisions will be resolved. Hence, for each bucket of the hash table we will have a linked list, initially empty.

A hash table will be protected by *multiple* locks to reduce lock contention while accessing the hash table from multiple threads. There will be one lock per M consecutive buckets in the hash table. We call such M consecutive buckets as a region. There will be $N/M = K$ regions, hence K locks. The first M consecutive buckets is the region 0 and is protected by lock 0; the next M buckets is region 1 and protected by lock 1, and so on. While doing an operation on the hash table (like insert, delete, get) and accessing a bucket, the corresponding lock must be acquired.

N can be a value between 100 and 1000. M can be a value between 10 and 1000. M should be $\leq N$ and N is a multiple of M. K can be a value between 1 and 100.

Your library will implement the following functions. It will define a `HashTable` type as well, so that applications can use this data type.

- **HashTable *hash_init (int N, int K)**. Creates a hash table of N buckets protected by K locks. Each bucket will have an associated linked list (chain) initialized to empty list. Returns a pointer to the hash table created if success; otherwise returns NULL.
- **int hash_insert (HashTable *hp, int k, void * v)**. Inserts key k and the associated value v into the hash table hp. Value is a pointer that can point to a structure containing data. The allocation and deallocation of memory for the structure is left to the application using the library. If the data is a simple integer, then it can be directly stored as the value (i.e., no need to separately allocate memory for the integer and store a pointer to that as the value). We are assuming `void *` and `integer` type has the same size. This function returns 0 if success; otherwise returns -1. If key already presents, does nothing and returns -1.
- **int hash_delete (HashTable *hp, int k)**. Removes key k and the associated value v from the hash table hp. If success returns 0, otherwise returns -1.
- **int hash_update (HashTable *hp, int k, void *v)**. Updates the value of key k to be v. If success returns 0, otherwise returns -1.
- **int hash_get (HashTable *hp, int k, void **vp)**. The value associated with key k is retrieved into a pointer variable (of type void *) whose address is vp. If success returns 0, otherwise -1.
- **int hash_destroy (HashTable *hp)**. Destroys the hash table and frees all resources used by it.

A multi-threaded application, for example x.c, that will use your library will first include the header file hash.h corresponding to your library. The application may create many threads and each thread may do a lot of table operations. An application will be compiled and linked with your library as follows:

```
gcc -Wall -o x -L. -lhash x.c
```

Develop a Multi-threaded Application

Implement a multi-threaded `integer-count` program. The program will take C text files as input. Each input file may contain a large number of positive integers. Repetitions are allowed. The program will count the number appearances of unique integers. In other words, for each integer appearing in one of those C files, it will find out how many times that integer occurs in those files. Your program will use a separate thread for each file. The threads will use a shared hash table to do counting. That means a key-value pair in the table will be an integer and its count. At the end, the program will write the integers in ascending sorted order to an output file. The output file will have a separate line for each unique integer. A line will contain an integer and its count separated by a colon as in the following output example:

```
17: 3
20: 1
```

```
145: 4
200: 2
500: 1
```

A sample invocation of the program is shown below:

```
integer-count 7 1.txt 2.txt 3.txt 4.txt 5.txt 6.txt 7.txt out.txt
```

There are 7 input files. The program will use 7 new threads. Each input file may contain an unsorted sequence of positive integers (keys). Each line of input will contain a single integer.

Experiments and Report

Assume we are wondering about the effect of the number of locks used (K) on the performance of the hash table operations. If there is one lock used, there will be high contention for it from multiple threads trying to access the hash table. Write an application (`test.c`) and design some experiments to analyze the effect of number of locks (K) on the performance. Performance metric can be, for example, the time to execute a set of operations on the hash table for a set of threads. Try to find out if you can see a performance increase while K is increased. Measure the time. Make a lot of experiments with different parameter values: number of threads (T), number of keys/operations (W), number of locks (K), table size (N). Try to see the effect of number of threads (T) on the performance as well. When T is 1, there will be no contention. Contention will increase when there are more keys be operated with and when there are more threads used. Show your numeric results in tables or graphs and try to interpret. Put your results, your interpretations, discussions into your report.

Submission

Put your report.pdf file and your program files (hash.c, integer-count.c, test.c) and a Makefile and a README.txt file into a directory named with your ID (for a group, a single file will be uploaded using the ID of one of the students). Then tar and gzip the directory. For example a student with ID 21404312 will create a directory named 21404312 and will put the files there. Then he will tar the directory (package the directory) as follows:

```
tar cvf 21404312.tar 21404312
```

Then he will gzip the tar file as follows:

```
gzip 21404312.tar
```

In this way he will obtain a file called 21404312.tar.gz. Then he will upload this file in Moodle.

Late submission will not be accepted (no exception). A late submission will get 0 automatically (you will not be able to argue it). Make sure you make a submission one day before the deadline. You can then overwrite it.

Tips and Clarifications

- You need to learn how to use Pthreads mutex variables. There are links to some resources in the References section of the course webpage. You can find additional resources from Internet.
- A project 2 skeleton of code is posted in github:
  `https://github.com/korpeoglu/cs342spring2019_p2`
  You can clone it and start with it. At least, it will help you with the compilation of the library and programs.
- You will include the Makefile with your project submission. Make sure it works in your environment (name your files accordingly). We will just type make and your programs should compile.
- We may put clarifications to the homepage of the course (near the project assignment).