CS342 Operating Systems - Fall 2016
Project 3: Synchronization, Mutex and Condition Variables

Assigned: Nov 10, 2016
Due date: Nov 24, 2016, 23:55

You can do this project in groups of two students each.

Objective: *Practicing synchronization, mutex and condition variables, multithreaded programming with Pthreads (POSIX threads), performing experiments and collecting measurement data, applying Probability and Statistics knowledge.*

PART A. Develop a library (libhash.a) that will implement thread-safe hash table data structure and 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.

A hash table has N buckets (buckets 0 through N-1). 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 is also an integer (it could be something else). 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 form multiple threads. There will be one lock per M consecutive buckets in the hash table. We call such an M consecutive buckets as a region. There will be N/M = K regions, hence K locks. The first M consecutive buckets is region 0 and protected by lock 0; the next M buckets is region 1 and protected by lock1, 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 <= 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 export also a HashTable type.

- **HashTable *hash_init (int N, int M).** Creates a hash table of N buckets protected by M 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 (Hash Table *hp, int k, int v).** Inserts key k and the associated value v into the hash table hp. If success returns 0, 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_get (HashTable *hp, int k, int *vptr).** The value associated with key k is retrieved into integer variable with address vptr. 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 app.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 app –llibhash.a app.c
```

We will develop test applications to test and stress your library implementation. You should do the same.

Experiments and Report. In this section, you will apply your knowledge from *Probability and Statistics* course. Do the following experiments and write your results into a report.

- Fix M. Change N. Measure the time to execute a multi-threaded program for various N values. Plot N versus running time. Try to fit a linear curve to this plot.
- For a fixed N and M, execute a multi-threaded program 100 times and find the mean and variance of program execution time.
- For a fixed N and M, execute a multi-threaded program with various number of threads such as 1, 2, 4, 8, 32, 64 and measure the time it takes to execute the program. Plot the number of threads versus running time. Try to fit a linear curve to this plot if possible. If not explain the reason.
- Fix N and change M so that K changes between 1 and 100 at increments of 10. For each K value (M value), for the same program, measure the time it takes to execute the program that is creating a lot of threads where these threads are doing a lot of table operations concurrently. Plot K versus time-to-execute the program. Try to fit a linear curve to this if possible. If not explain the reason.

Part B. A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds the TA currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back at a later time.

Using POSIX threads, mutex locks and condition variables implement a solution (write a program called ta.c) that coordinates the activities of the TA and the students.

Using Pthreads, begin by creating N students. Each will run as a separate thread. The TA will run as a separate thread as well. Student threads will alternate between programming for a period of time and seeking help from the TA. If the TA is available, they will obtain help. Otherwise, they will either sit in a chair in the hallway or, if no chairs are available, will resume programming and will seek help at a later time. If a student arrives and notices that the TA is sleeping, the student must notify the TA. When the TA finishes helping a student, the TA must check to see if there are students waiting for help in the hallway. If so, the TA must help each of these students in turn. If no students are present, the TA may return to napping.

Perhaps the best option for simulating students programming—as well as the TA providing help to a student—is to have the appropriate threads sleep for a random period of time.

We will invoke your program as follows:

```
ta
```

Clarifications:

- You need to learn how to use Pthreads mutex and condition variables. There are links to some resources in the References section of the course webpage. You can find additional resources from Internet.
- We will post sample skeletons of code. You can download and study it. 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 will compile your programs.
- We may put clarifications to the homepage of the course (near the project assignment).