COSC 301 Project 2, Fall 2012: Multithreaded Hashtable Library

Due Friday, October 26, 2012

Hashtables are very widely used data structures. For your second project, you'll write a multithreaded hashtable library. Your library will have some similarities to the linked-list library that you modified to be threadsafe in lab 5, but will, of course, implement a hashtable instead of a linked list. A further difference is that your hashtable will need to store strings rather than integers.

If you want to refresh your memory on what a hashtable is, you can take a look at the Wikipedia entry: <`http://en.wikipedia.org/wiki/Hash_table`>. A brief description of one way to implement a hashtable in C is also described below.

You'll want to put all your code in a `git` repository, since you'll turn in your project the same way you turned in the code for the shell project.

As with the first project, you can (and are encouraged to) collaborate in groups of up to 2 persons. Only one submission needs to be made per group, but please note as a comment in your code who worked on it and how the work was broken down.

Project description

The main requirements for this project are to:

1. Implement each of the 5 library functions identified in `hash.h` (and shown below).
2. Make the library threadsafe (i.e., you'll need to create one or more mutexes to ensure mutual exclusivity inside the various library functions).
3. Do your best to maximize concurrency. You want to permit as many threads as reasonably possible to perform simultaneous operations on the hashtable. I'd recommend that you first implement the library *correctly*, then optimize for concurrency.

There's a bit of starter code posted in the class git repo. You are given a `hash.h` file that defines the API that your library must support. It includes the following functions (these are copied from `hash.h`):

```
// create a new hashtable; parameter is a size hint
hashtable_t *hashtable_new(int);

// free anything allocated by the hashtable library
void hashtable_free(hashtable_t *);

// add a new string to the hashtable
void hashtable_add(hashtable_t *, const char *);

// remove a string from the hashtable; if the string
// doesn't exist in the hashtable, do nothing
void hashtable_remove(hashtable_t *, const char *);

// print the contents of the hashtable
void hashtable_print(hashtable_t *);
```

You can decide how the `hashtable_t` type is defined. An empty definition is already present in `hash.h`, and you can add to it any way you like.

Also included in your starter files are `main.c`, `words.txt`, and a `Makefile`. The code in `main.c` is a program that exercises the functions in your hashtable library. It can create a specified number of threads to add a specified number of strings to the hashtable (note that reads in the contents of `words.txt` and adds strings from that file to the hashtable). When you run `make`, the program `mthash` will be created by compiling `main.c` along with your hashtable library files. If you type `mthash -h` you'll get the following help text:

```
usage: ./mthash [-h] [-t threads] [-m max_values_to_add] [-s num_hash_buckets]
    -m: num of values for each thread to add to hashtable
    -s: number of hashtable buckets
    -t: number of threads to start up
    -h: show this help
```

You can use these command-line options for creating any number of threads to add and remove items to/from the hashtable, specifying the number of strings that each thread should add and remove from the table, and for specifying a size hint to the hashtable library when creating a new hashtable.

A basic hashtable in C

The most straightforward way to implement a hashtable in C is to create an array of linked lists (this is referred to as "separate chaining" on the Wikipedia page for hashtables, though the term I know it as is "linear bucket chaining"). Each entry in the array is often referred to as a hash "bucket". You'll need to create a hash function that takes a string and produces an integer, which maps to one of the buckets. You can then manipulate the associated linked list to add or remove an item.

You'll notice that the `hashtable_new` library function takes one integer as a parameter. This should be considered as a *hint* for how many buckets to create in your hashtable. You aren't required to create exactly that many buckets; treat the parameter as a hint rather than a demand. The value that gets passed to your `hashtable_new` function can be specified on the command line to `mthash` by using the `-s` option (see above).

You should try to create a *good* hash function, which is one that should (as much as possible) uniformly distribute data across the array of buckets. A little bit of web searching should lead you to something reasonable.

How to turn in

Just post a link to your `git` repository on Moodle, just like with project 1.