CSE 303, Spring 2007, Assignment 7
Due: Wednesday 30 May, 9:00AM

Last updated: May 21

Summary: This homework involves "string sets" much like the ones you implemented in homework 4. Starting with C code provided to you, you will make the library thread-safe (problem 1), demonstrate the original version is not thread-safe (problem 2), port the single-threaded version to C++ (problem 3), and compare your C++ code to an alternate implementation (problem 4).

1. Copy the provided file `string_set_st.c` (`st` for single-threaded) to `string_set_mt.c` (`mt` for multi-threaded). Make the functions in `string_set_mt.c` correct even if called from multiple threads in parallel. The string-set should correctly identify what has been added and should never have two entries that are equal strings. Do so as follows:

- Do not modify `string_set.h`.
- Change the definition of `struct StringSet` so each string-set has a lock and a count of how many "clients are using" the string-set. This *use-count* is explained more below. Change `new_string_set` to initialize the lock and use-count (the use-count should start 1).
- Change the other functions (except `destroy_string_set`) to acquire the argument's lock before using any other fields and release the lock just before returning. To avoid ever reacquiring an already-held lock, you should create one or more helper functions and reorganize the code a bit.
- `use_string_set` should increment the string-set's use-count (unless it is already negative; see below). It is up to client threads to call this function appropriately; i.e., it is not your problem if it is called the wrong number of times.
- `finished_string_set` should decrement the string-set's use-count. If and only if doing so makes the count 0, it should then deallocate the strings and the array in the set, but not the set itself. It should also set the use-count to -1 in this case. Once the use-count is -1:
  - It should remain negative forever.
  - Any call to `use_string_set`, `string_set_member`, `string_set_add`, or `string_set_foreach` with *this string-set* should return -1 without accessing any fields except the lock and use-count. (This -1 indicates an error; a finished string-set should not be used anymore. However, we do not destroy the string-set in case other threads do try to use it.)
- `destroy_string_set` does actually destroy the string-set's lock and deallocate the string-set itself. It should not use any synchronization (we assume the caller is correct that no further or parallel access can exist) and it should assume the array and strings have already been deallocated.

Sample solution has about 50 new lines of code.

2. Change `main.c`, `string_set_st.c`, and `string_set_mt.c` to demonstrate that the original code has a race-condition that your version does not. Do so as follows:

- In `main.c` add code so the assertion in `add_and_get` can fail for the original code (proving that a string added to a set is not in the set). In particular, create two threads (and then join on them so that the main thread does not exit until they are done). Adjusting use-counts is unnecessary.
- In `string_set_st.c` and `string_set_mt.c`, choose *one* place to add a call to library function `sched_yield()`. (This function pre-empts the calling thread, so if you pick a good place, the assertion failure should happen quite naturally, i.e., pretty much every time.) Choose the *same* conceptual place in both files.
- Space leaks in `main.c` are fine (i.e., you do not need to deallocate the string-set).

Sample solution has 6 new lines of code.
A Makefile has been provided. After adding your code, running `okay` should terminate normally and running `broken` should lead to a failed assertion.

3. The provided files `string_set_classes.h` and `string_set_classes.cc` define an abstract class `StringSet` and a subclass implementing string-sets with linked lists. You should not change any of this code. Instead, provide a second subclass of `StringSet` that implements string-sets with a resizing array of `string` objects. In other words, "port" the *single-threaded* C code provided to you to C++ using an OOP style. More specifically:

- In the header file, define the class, with public methods and private fields. Put all method definitions in the `.cc` file.
- Define a constructor and destructor. Your destructor should delete the underlying array. Because the array holds objects, not pointers, you should not need to do anything else; deleting the array should invoke the destructors for each array element.
- The only method of `string` you need to know about is `compare` and it is also used in the linked-list implementation given to you.
- To resize the array, use new to make a bigger array of string objects and be sure to deallocate the old array.
- Notice the `foreach` method takes a `StringDoer` object instead of a function pointer.
- In general, change the code to use classes and string objects, but do not change the basic algorithm.

Sample solution has about 45 new lines of code.

4. (This problem requires you to write English, not code.) After completing problem 3, the provided file `string_set_test.cc` should run correctly.

1. In one English paragraph not to exceed about half a page, explain what this test file does. Be precise, discussing the six different tests, what sort of string-sets are created, what is put in them, etc.
2. In a second English paragraph and a small table, present the running time for the six different tests and explain exactly *why* some of the tests are significantly faster than the others.

Put your answer in a text file called `problem4`.

Extra Credit: Do one or more of the following.

- In C, make a version of the string-set library that is thread-safe and shares equal strings across string-sets. Store with each string (use a struct instead of just a `char*`) a reference-count (indicating how many not-finished sets are using the string) and free a string only when the reference-count is zero.
- In C, write a client of the string-set library that can deadlock without creating any locks other than the ones created by creating string-sets. This is difficult because the locks are hidden from the client, but it is possible using global variables.
- In C++, write a string-set client that uses the `foreach` method to calculate the sum of the lengths of all strings in a set. In a text file, explain why the C code's `foreach` is not good for this task. Finally, extend the C interface with a function `foreach_better` that is appropriate for this task yet as generic as the C++ code (hint: use `void*`) and implement the string-set client in C using this new function.

Turn-in: Use the `turnin` command (man turnin) for course cse303 and project hw7. Given the large number of files, it is easiest to run turn-in on a whole directory *after* running make clean and removing any temporary files. For example:

```
turnin -ccse303 -phw7 myhw7
```

If you use one late-day (see the syllabus) use the project hw7late1 instead of hw7 and similarly hw7late2 for two late days. If you do the extra credit, turn in extra files as necessary *and turn in a text file titled* `extra_credit` *that explains what extra credit you did.*