CS520 FALL 2019

LAB 8

Description

Update your symbol table facility from Lab 5 to support concurrent (multithreaded) access to the created table.

Complete this assignment by doing the following steps in the order given:

1. Copy *symtab.h* from *~cs520/public/lab8*. **Read it carefully!**
2. A second parameter has been added to the *symtabCreate* function to indicate the number of POSIX mutexes to use to protect access to the table. Modify your code to include this parameter. This parameter needs to be at least 1, and should be stored in the control struct for the new table. Also, you need to malloc space for an array of the mutexes and store the base address of that array in the control struct. Finally, scale the size hint provided as the first parameter so that the size of the table that is created is the closest even multiple of the number of mutexes.
3. A third parameter has been added to the *symtabCreate* function to indicate whether thin locks should be used instead of POSIX mutexes to protect access to the table. Modify your code to include this parameter. This parameter should be stored in the control struct for the new table. However, for this lab, if the third parameter is not 0, print an error message to stderr and exit the program.
4. The *symtabInstall* function has been replaced by the *symtabUpdate* function. With the old interface, users would often follow a *symtabLookup* call with a call to *symtabInstall*, in order to modify the data associated with a symbol already installed in the table. With the new interface, this can now be done with one call to *symtabUpdate*. The trick is that the caller now passes a callback function to *symtabUpdate*, which is called with the old data pointer for the given symbol. The user can then examine and modify the old data and return an updated data pointer back to *symtabUpdate* as the callback function's return value. *symtabUpdate* then installs the updated data pointer with the symbol in the table. *symtabUpdate* takes a fourth argument that is passed to the callback function as its second argument. Modify your code for *symtabInstall* to implement *symtabUpdate*.
5. Groups of hash buckets will be assigned to mutexes by taking the hash index of the bucket modulo the number of mutexes specified when the table was created. When *symtabLookup* or *symtabUpdate* are called for a table, lock the mutex for the symbol's bucket upon entry to the function and unlock that mutex right before exit of the function.
6. To test your symbol table implementation, you need to modify the main.c from Lab 5 for threaded support. For the purposes of this lab you need to supply just one file name as the argument to your main function i.e. you are going to be testing your implementation for a single thread only. This is a concurrent version of the program we used to test Lab 5. You will be submitting only your lab8.c file containing the symbol table implementation and we will test it by using our main.c file.

   To be clear your main.c file will be responsible for doing the following (these are comments on top of the main.c file Lab 5):

```
//
// This program reads a series of filenames from the command line.
// It opens and reads each file to count how many times each word
// appears in the collection of files.
//
// A word starts with a letter (either uppercase or lowercase) and
// continues until a non-letter (or EOF) is encountered. The program
// ignores non-words, words shorter than six characters in length,
// and words longer than fifty characters in length. All letters
// are converted to lowercase.
//
// After reading all the files, the program then prints the twenty
// words with the highest counts.
//
```