Megan Sullivan
HW1 – Multithreaded Programming

Read-Write Locking

Here are a list of the mutexes and global variables I created to implement the read-write locking system:

request_mutex: Having possession of this lock indicates that you are the thread whose request is currently being processed by the database.

num_readers: This integer indicates how many readers are currently in the database.

num_readers_mutex: Since num_readers will be accessed by multiple threads, this mutex is to protect that shared variable.

db_mutex: This is a lock for the entire database.

There are three commands that manipulate the database directly: query, add, and xremove. Query is strictly a reading command, while add and xremove are both writing commands.

First let's examine the reader case. The first thing a reader does is acquire the request_mutex. Once you proceed from this point, you know you are the only command currently being processed by the database. Next the reader acquires the lock to num_readers and increments it (since you are a reader and you are now in the database). Now the reader checks to see if num_readers is equal to one. If it is, this means you are the first reader in the database, so you should acquire the lock to the database. This will prevent any writers from being able to get into the database while you are still reading. Then you release the lock on num_readers and the request lock so that other requests can be processed. Now you are able to do your actual query on the database. Once you are finished reading, you reacquire the num_readers_mutex and decrement it (effectively taking yourself out of the database). Then you need to check to see if num_readers is equal to zero. If it does, that means you are the last reader leaving the database, so you should release the db_mutex so that writers (or other readers) can get into the database next.

Because you only attempt to acquire the db_mutex when you are the first reader in the database, other readers after the first reader will still be able to enter the database and perform their queries. Any writers attempting to get in will block, because they cannot acquire the db_mutex. And since reading from the database takes a constant amount of time and will always complete, you are guaranteed that eventually all the readers will finish their queries and the last one out will free up the db_mutex so that other threads may enter.

Now let's examine the writer case (for the add and xremove commands). This one is much simpler. First you acquire the request lock, so that you can ensure that you are the only thread whose command is being processed by the database. Then you acquire the db_mutex. As previously stated, you will not be able to successfully acquire this lock until it all readers who may be in the tree have exited. Then once you have this lock (and therefore access to the database), you can release the request_lock (so that other threads may get in line to be processed), perform your write, and then release the db_mutex so that other threads can work with the database.