Jonathan Chu
CSCI-402
February 22, 2013

Part D: Fine

Design and Implementation

To achieve having reader and writer functionality on a per-node basis, I added a lock to the node struct in the db.h file. Icne I had done a full implementation of reader and writer locks, I decided to use the pthread_rwlock this time to make my code more readable. In the node_create I initialize this rwlock, and in the node_destroy function, I destroy the lock. Afterwards, I do the same locking as RW in the functions, but on a per-node basis. I do this by locking the parent of the node being operated on (regardless of it being query, add, or xremove), then locking the node (in the case of add and query). This ensures that other threads that are operating on the database will not make changes on nodes related to the node being operated on.

For add, I write lock the node under which the new entry will be added. For query, I read lock the parent and node to be read. Finally, in xremove, I write lock the node being operated on, and I read lock and eventually release the nodes that I traverse over to find the replacement node of the one to be deleted. This in effect locks the parent of the node to be deleted. After some testing, I realized that my locks were not tight enough, and allowed for race conditions to happen. To prevent this, I did parent locking for each of the functions in the search function (I ended up creating 2 different searches, one for query and the other for add and xremove). As I search for the node to be operated on, I read lock and eventually release the nodes I traverse over.

Known Defects

There are no defects that I know of in the code.