UNIVERSITY OF SOUTHERN CALIFORNIA

CS402 Assignment 01 – Read/Write Locking

Michael Gerow

February 18, 2013

1 Read Write Locking

Read write locking is implemented similarly to coarse grained locking in that we only have on global lock for the entire table, but instaed of using a pthread_mutex_t we instead use a pthread_rwlock_t. This allows us to use the functions pthread_rwlock_rdlock in order to get a read lock and pthread_rwlock_wrlock in order to get a write lock.
There is also one small difference with the way we compile under linux (which can be seen in the if (!initialized) clause within interpret_command function). The POSIX spec does not mentiona anything about giving readers or writers priority. Because of that, a lot of the implementations seem to not really care about priority and just allow any waking thread to continue through the lock, as long as it is allowed. This has the unfortunate effect of basically allowing readers to continue pouring through a lock even if a writer is waiting to acquire it. In order to solve this we enable the attribute PTHREAD_RWLOCK_PREFER_WRITER_NONRECURSIVE_NP when we are compiling under linux (which supports this attribute). By doing this we should see an increase in performance when we have a relatively even number of readers and writers.

1.1 Testing

We test the correctness of the read write lock similarly to the corarse grained lock, using the test that adds a lot of elements to the database and then removes them, which should leave us with an empty database. This can, like with coarse, be done simply by running:

```
./server_rw < test/250_add_remove | tail -n 800 | less
```

Of course, we should also expect an increase in speed for read write locking. It turns out that this is only true for certain types of accesses of the database. Performance testing will be covered in the performance document.