CS6240 Homework 5

Name: Wenqing Xu

1) Case 1:

a. Observe that you hit an assert statement as readers and writers conflict. Explain why.

If I do not set NOASSERT=1, the assert will fail with statement "Assertion failed: (num_writers == 1 && num_readers == 0), function begin_write, file read_write_locks.c, line 122." Because I do not put any locks inside the functions, all of the 4 threads will run in parallel, which certainly violates the Assertion (num_writers == 1 && num_readers == 0).

b.

If I set NOASSERT=1, I will get the following output :

*** 20 reads and 20 writes in 116 milliseconds

Each reader or writer has 10 tasks and each task sleeps for 10 millisecond. Now that we see the program takes around 100 milliseconds, which tells us that all the threads run in parallel because there is no locks.

2) Case 2:

The output of case2 is:

*** 20 reads and 20 writes in 459 milliseconds

2 Readers and 2 writers have 40 tasks in total. We can see that the running time is about 400. So threads run one-by-one in this case.

3) Case3:

a. The code in 3.a. takes a very long time because the pthread_mutex_unlock() is after the nanosleep(). The reason of that is the program "holds the lock while sleeping". The function will sleep with the lock and the continue looping and probably sleep again. Without unlocking the lock, other threads cannot break in and start running. So the 3.a. code is very slow.

b. After fixing the code, the lock is released before sleeping and other threads can start working. The bug goes away.

the output is:

*** 20 reads and 20 writes in 348 milliseconds

The running time is about 300 milliseconds because readers are using share lock so readers can run in parallel; writers using exclusive locks run sequentially. Writers take 200 milliseconds, 100 each.

4) Case 4:

a. output:

*** 20 reads and 20 writes in 316 milliseconds

The writers hold exclusive lock one-by-one and they run sequentially. The readers use pthread_mutex_trylock() function to test whether the write lock is held. If not, the reader starts to run in parallel.

2 writers run sequentially and 2 readers run parallelly, making the running time about 300 milliseconds.

b. output:

*** 50 reads and 20 writes in 315 milliseconds

When increase the number of readers, the running time remains the same because readers run in parallel. In other words, the reading tasks will take 100 milliseconds as long as there are enough CPUs to use.

c. output:

*** 50 reads and 50 writes in 631 milliseconds

When increase the number of writers, the running time increase by 100 milliseconds for each additional writer. For the reason that writers need to acquire an exclusive lock and release it one after another. Each writer will take another 100 milliseconds in the running time.

d.

running time = 100 + 100 * num_writer + t

t is a symbol representing a small number of time because there is so other

operations consuming some time. Also, there is the possibility that the writer cuts off between, say first half and the other half readers. In such scenario, the reader will take more than 100 milliseconds. However, when I run the program using the login.ccs.neu.edu machine, I never see such scenario happening. In conclusion, the formula takes 100 for reading and 100*num_writer for writing and t for other things.