CS 6240 - Large-scale Parallel Data Processing
Homework 5
Name: Srikanth Babu Mandru

Question 1.a:

Since one writer or reader has already started execution and haven't yet released the lock that it acquires or haven't completed its task, this makes other writer (started execution at the same time) to raise an error preventing it from acquiring the lock and executing "begin_write" function (programmatically, num_writers will be more than 1 or num_readers will be more than 0, which causes an assertion here).

Question 1.b:

Output:

Code has completed with following:

"*** 20 reads and 20 writes in 116 milliseconds"

Explanation:

Even though the code ran without errors for 2 readers and 2 writers, only few tasks were executed before hitting the assertion statement and finishing the code run. Hence, we see time about 100 milliseconds. That is, around 10 tasks were executed (since each task takes 10 milliseconds) either by reader or writer that acquires the lock first.

Question 2:

Explanation:

20 reads and 20 writes were performed, that is 40 tasks. But, given that each task takes about 10 milliseconds. Thus, the total time taken was,

Total time = 40 * 10 milliseconds + wait time for acquiring and release of locks
= 478 milliseconds (result from my code run)

Question 3.a:

Explanation:

In short, answer is "deadlock". This was caused because the second writer has been waiting for long time to acquire the lock that was actually being used by the first writer and wasn't released.

Output:

I got an Alarm clock as follows due to the infinite loop execution of code:

```
make[1]: *** [check] Alarm clock: 14

make: *** [case3] Error 2
```

Question 3.b:

Explanation:

By moving the "pthread_mutex_unlock" before the nanosleep, we are actually releasing the lock acquired by first writer. This makes the second writer to acquire the lock and do its processes. Conceptually, we have prevented the application program to get into the "Deadlock" state.

Results are as follows:

*** 20 reads and 20 writes in 381 milliseconds

Question 4.a:

Output:

*** 20 reads and 20 writes in 463 milliseconds

Explanation:

Now, the second writer waits till the lock is released by first writer. This wait time can be seen in the time mentioned above. May be second writer has waited for around 60 milliseconds before starting its tasks.

Role of locks:

Locks make the resources to be acquired/released based on the conditions that we have provided in the code so that resources are optimally used without causing any errors or inconsistencies while doing parallel jobs or operations on same resource.

Question 4.b:

Output:

*** 50 reads and 20 writes in 768 milliseconds

Explanation:

Now, there are 70 tasks in total that is 50 reads (due to 5 readers) and 20 writes (due to 2 writers). Each task takes 10 milliseconds to run.

Thus, total time = 70 * 10 + wait times due to concurrent writers = 768 ms

Question 4.c:

Output:

*** 50 reads and 50 writes in 1146 milliseconds

Explanation:

Now, there are 100 tasks in total that is 50 reads (due to 5 readers) and 50 writes (due to 5 writers). Each task takes 10 milliseconds to run.

Thus, total time = 100 * 10 + wait times due to concurrent writers

Question 4.d:

Formula,

Total time = (num_writers * num_tasks * task_time) +
(num_readers * num_tasks * task_time) + wait-time

Where, task_time – “Time taken by each task to finish”