Tutorial 6 Answer Sheet

CS3103 Operating Systems

Student Name: LIU Hengche

Student No. : 5 7 8 5 4 3 2 9

Day: ☐Tuesday ☑Wednesday ☐Friday

Time: ☐ 10:00 - 10:50 ☐ 11:00 - 11:50 ☐ 16:00 - 16:50 ☑ 18:00 - 18:50

Deadlocks

Submission:

- Deadline: Sunday, April 7, 2024, 23:59 pm HKT.
- Answers are allowed in text only. Any form of image/snapshot is not allowed.
- Submit this answer sheet via Canvas->Files->Tutorials->Tutorial 6.

Questions

Question 1: First let's make sure you understand how the programs generally work, and some of the key options. Study the code in `vector-deadlock.c`, as well as in `main-common.c` and related files. Now, run `./vector-deadlock -d -n 2 -l 1 -t`, which instantiates two threads (`-n 2`), each of which does one vector add (`-l 1`). Change the number of loops (`-l`) from 1 to higher numbers. What happens? Does the code (always) deadlock?

Answer:

When the number is relatively small, the result would be "Time: 0.00 seconds." But as the number of loops increases to a relatively large number like 1000, I could not get the result. This suggests that the code does not always deadlock.

Question 2: How does changing the number of threads (`-n`) change the outcome of the program? Are there any values of `-n` that ensure no deadlock occurs?

Answer:

When the number of threads increases, it is more likely to have deadlock. For example, the command "./vector-deadlock -d -n 2 -l 200 -t" gives the result of "Time: 0.00 seconds.", but when we increase the threads to the amount of 50, i.e. "./vector-deadlock -d -n 50 -l 200 -t", it could not give us a result, which suggests deadlock occurred. So increasing the number of threads would make it more likely to have deadlocks.

Yes. If we set the number of threads to be 1, then there would never be deadlocks. So "-n 1".

Question 3: Now examine the code in `vector-global-order.c`. First, make sure you understand what the code is trying to do; do you understand why the code avoids deadlock? In the `vector_add()` routine, there are two cases to consider: when the source and destination vectors are different and when they are the same. What is the difference between these two scenarios, i.e., why we should consider the later one?

Answer:

The reason that the code avoids deadlock is because this version of vector_add() grabs the locks in a total order, based on the address of the vector.

The difference is that if the source and destination vectors are the same then we only need one mutex lock. If we don't give this case special consideration, then the code would try to acquire a lock that is already held, thus causing deadlock.

Question 4: Now run the code with the following flags: `-t -n 2 -l 100000 -d`. How long does the code take to complete? How does the total time change when you increase the number of loops, or the number of threads?

Answer:

The output is as follows:

hengchliu2@ubt20a:~/tutorial6$ ./vector-global-order -t -n 2 -l 100000 -d

Time: 0.06 seconds So it takes 0.06 seconds for my case.

When we increase the number of loops, the total time increases (Running with the flags "-t -n 2 -l 200000 -d" would take the total time of 0.11 seconds). The relation is roughly linear.

When we increase the number of threads, the total time also increases(Running with the flags "-t -n 4 -l 100000 -d" would take the total time of 0.17 seconds). The relation is also roughly linear.

Question 5: What happens if you turn on the parallelism flag (`-p`)? How much will the performance (i.e., total time) change when each thread is working on adding different vectors (which is what `-p` enables) versus working on the same ones?

Answer:

When we turn on the parallelism flag (-p), the program will execute faster.

"-t -n 2 -l 100000 -d" would take roughly 0.06 seconds, when we turn on the "-p" flag, then total time becomes 0.03 seconds, so it will save 0.03 seconds for the case where total threads number is 2. It saves more as we increase the number of threads. (Around 0.04 seconds for 3 threads, around 0.17 seconds for 5 threads.)

Question 6: Now let's study `vector-try-wait.c`. First make sure you understand the code. Is the first call to `pthread_mutex_trylock()` really needed? Now run the code. How fast does it run compared to the global order approach? How does the number of retries, as counted by the code, change as the number of threads increases?

Answer:

Yes, the call to the function is needed. This is because the code uses pthread_mutex_trylock() to attempt to grab locks, when the try fails, it simply releases the locks that it's currently holding and return to the top to try again.

Compared to the global order approach, this version of vector_add() runs slower. For the command of "-t -n 2 -l 100000 -d", the global order approach takes 0.06 seconds, while retry approach takes around 0.18 seconds.

For "-t -n 2 -l 100000 -d", total number of retries is 549535:

hengchliu2@ubt20a:~/tutorial6$ ./vector-try-wait -t -n 2 -l 100000 -d

Retries: 549535

Time: 0.18 seconds;

For “-t -n 3 -l 100000 -d”, total number of retries is 892139:

hengchliu2@ubt20a:~/tutorial6$ ./vector-try-wait -t -n 3 -l 100000 -d

Retries: 892139

Time: 0.35 seconds

For “-t -n 5 -l 100000 -d”, total number of retries is 2960624:

hengchliu2@ubt20a:~/tutorial6$ ./vector-try-wait -t -n 5 -l 100000 -d

Retries: 2771758

Time: 1.52 seconds

From above we can see that the number of retries increases dramatically as we increase the number of threads.

Question 7: Now let’s look at `vector-avoid-hold-and-wait.c`. What is the main problem with this approach? How does its performance compare to the other versions (i.e., `vector-global-order` and `vector-try-wait`), when running both with `-p` and without it?

Answer:

The main problem is low efficiency and long running time. From the code we can see that there is a global lock that protects all lock acquisition, which suggests that even if the vectors will not enter deadlock, the global lock still requires vectors to enter the critical section one at a time.

The performance for global mutex is less satisfactory comparing to the global order, but better than retry. For the command with flags “-t -n 2 -l 100000 -d”, it will take 0.10 seconds to execute; for “-t -n 3 -l 100000 -d”, it will take 0.26 seconds; for “-t -n 5 -l 100000 -d”, it will take 0.51 seconds.

However, when we run the code with -p flag, “-t -n 2 -l 100000 -d” takes 0.05 seconds; “-t -n 3 -l 100000 -d” takes 0.08 seconds; “-t -n 5 -l 100000 -d” takes 0.14 seconds. This performance is close to that of global order.

Question 8: Finally, let's look at `vector-nolock.c`. This version doesn't use locks at all; does it provide the exact same semantics as the other versions? Why or why not?

Answer:

No, the semantic for atomic operations is that for a pair it is mutually exclusive, it does not include that all are mutually exclusive like mutex lock. For mutex lock, the semantic is that all are mutually exclusive since it could only be the one holding the locks that executes vector_add() function.

Question 9: Now compare its performance to the version `vector-global-order`, both when threads are working on the same two vectors (no `-p`) and when each thread is working on separate vectors (`-p`). How does this no-lock version perform?

Answer:

For "-t -n 2 -l 100000 -d":

hengchliu2@ubt20a:~/tutorial6$ ./vector-nolock -t -n 2 -l 100000 -d

Time: 0.55 seconds

The execution time is 0.55 seconds;

For "-t -n 3 -l 100000 -d":

hengchliu2@ubt20a:~/tutorial6$ ./vector-nolock -t -n 3 -l 100000 -d

Time: 0.60 seconds

The execution time is 0.60 seconds;

For "-t -n 5 -l 100000 -d":

hengchliu2@ubt20a:~/tutorial6$ ./vector-nolock -t -n 5 -l 100000 -d

Time: 1.21 seconds

The execution time is 1.21 seconds.

Without flag "-p", the performance is worse than global order version.

For “-t -n 2 -l 100000 -d -p”:

hengchliu2@ubt20a:~/tutorial6$ ./vector-global-order -t -n 2 -l 100000 -d -p

Time: 0.02 seconds

The execution time is 0.02 seconds;

For “-t -n 3 -l 100000 -d -p”:

hengchliu2@ubt20a:~/tutorial6$ ./vector-global-order -t -n 3 -l 100000 -d -p

Time: 0.04 seconds

The execution time is 0.04 seconds;

For “-t -n 5 -l 100000 -d -p”:

hengchliu2@ubt20a:~/tutorial6$ ./vector-global-order -t -n 5 -l 100000 -d -p

Time: 0.04 seconds

The execution time is 0.04 seconds.

Without flag “-p”, the performance is better than global order version.