Comp 310 Assignment 2 - Reader-Writer Problem
Michel Abdel Nour - 260725050

Running the program

In order to run the program, type the following commands in terminal after accessing the A2 folder:

$ gcc -pthread A2Q1.c
$ ./a.out NUMBER_OF_WRITES NUMBER_OF_READS

Question 1

We run the program using the default parameter values: NUMBER_OF_WRITES = 30, NUMBER_OF_READS = 60

We get the following thread statistics:

Read minimum wait time (in microseconds) : 0.000000
Read maximum wait time (in microseconds) : 0.991000
Read average wait time (in microseconds) : 0.012694

Write minimum wait time in microseconds : 0.007000
Write maximum wait time in microseconds : 0.146000
Write average wait time in microseconds : 0.010737

Ratio (avg write wait time/avg read wait time) : 0.845804

We notice in this case the write wait time is very close to the read wait time and thus there is no starvation problem that is visible in this case.

Question 2

By choosing new parameters where we increase the ratio of readers to writers and changing the sleep times in the threads to microseconds:

NUMBER_OF_WRITES = 1, NUMBER_OF_READS = 1000
Using the function usleep() that takes arguments as microseconds:

We get the following thread statistics:

Read minimum wait time (in microseconds) : 0.000000
Read maximum wait time (in microseconds) : 65.756000
Read average wait time (in microseconds) : 0.028329

Write minimum wait time in microseconds : 0.007000
Write maximum wait time in microseconds : 142.538000
Write average wait time in microseconds : 63.491000

Ratio (avg write wait time/avg read wait time) : 2241.227070

Now the starvation problem appears to be very significant given a larger number of read operations. This is because readers are allowed to read shared resources concurrently and the writers don't gain access to the shared data until the last reader releases the lock on rw_mutex. Moreover, by decreasing the sleep times, the overlap of readers and writers attempting to execute is more frequent and probable, which causes the writers, which does not have priority in the implementation, to wait longer than the readers.

For the implementation of the Reader-Writer Problem in "A2Q1.c" using pthreads, we conclude that writers are starved given that the average waiting times for writers are significantly larger than the average waiting times for readers for larger readers to writers ratios and shorter sleeping times.

Question 3

To solve the writer starvation problem, a modified version of the implementation in "A2Q1.c" located in "A2Q3.c" uses an additional semaphore in order to ensure fairness in waiting times for both readers and writers and prevent starvation.

The added semaphore "queue_mutex" is locked in the beginning of the reader / writer functions and released after the reading and writing operations have been executed. This method resembles a FIFO queue where the readers and writers are not given priority over one another.

Question 4

For this part, we need to accentuate the starvation problem in order to measure the effectiveness of the new implementation. Therefore, we keep the sleep times smaller and in microseconds.

We run the modified implementation with the default parameter values: NUMBER_OF_WRITES = 30, NUMBER_OF_READS = 60

We get the following thread statistics:

Read minimum wait time (in microseconds) : 0.000000
Read maximum wait time (in microseconds) : 265.730000
Read average wait time (in microseconds) : 0.095654

Write minimum wait time in microseconds : 0.000000
Write maximum wait time in microseconds : 0.325000
Write average wait time in microseconds : 0.002553

Ratio (avg write wait time/avg read wait time) : 0.485471

We notice that the waiting time ratio is smaller here. Therefore, the starvation problem is minimized.

We try again with the modified implementations, now setting the parameters to: NUMBER_OF_WRITES = 1, NUMBER_OF_READS = 1000

We get the following thread statistics:

Read minimum wait time (in microseconds) : 0.000000
Read maximum wait time (in microseconds) : 265.730000
Read average wait time (in microseconds) : 0.095654

Write minimum wait time in microseconds : 0.007000
Write maximum wait time in microseconds : 63.904000
Write average wait time in microseconds : 19.317200

Ratio (avg write wait time/avg read wait time) : 201.948889

We notice the ratio of wait times decreased 10-folds compared to the run (1, 1000) in Q2. The new implementation is therefore effective at reducing the writer starvation problem.

IMPORTANT NOTE

I tested with different sleeping times, and the conclusion was that having smaller sleep times inside the threads led to better (smaller values) results in terms of the ratio of average write wait times over average read wait times.

By testing with sleep times of microseconds, the starvation problem is a lot more visible.