CMPSC 140 Summer 2022: Homework 4: Pthreads

Instructor: Richard Boone

Due: Monday August 1st, 2022 9 PM

Purpose: This assignment is meant to familiarize you with programming using Pthreads, as well as a number of methods by which you can deal with the conventional issues caused by multi-threaded programs.

Written Questions:

1. (5 points) The producer-consumer threads can be synchronized using a semaphore as follows:
   Consumer thread:
   sem_wait(s);
   Consume an item
   Producer thread:
   Produce an item
   sem_post(s);

   Describe 3 application cases in which it makes sense to set the initial semaphore value s as 0, positive value, or negative value, respectively. Notice that pthread semaphore implementation does not allow a negative initial value. You can refer the semaphore definition discussed in class on the impact of having a negative value.

2. (5 points) Book problem 4.12

Programming:

1. (20 points) Attached to this handout is a C file, "histogram.c" for generating a histogram of random data using serial programming. Test out the file a few times to see how it works. You will alter this file to work in parallel using pthreads. Program requirements are as follows:

   (a) You should parallelize the for loop which assigns pieces of data to the histogram. For this program, no parallelization of the data creation or bin creation is necessary.

(b) Using a timer and a barrier, calculate the total time taken to generate the histogram. Print out the total time taken when the program finishes. Ensure you're careful with where you time as you only want to measure the time of the parallelized portion.

(c) Use mutexes to control access to the histogram. You should have two versions, one uses a single mutex to control access to the entire histogram. The other uses a mutex per bin. All testing should be done with both versions of the histogram.

(d) At large numbers of samples, the print function reaches across multiple lines per bin. Alter the print function so that the bin with the most values prints out 100 "X" characters, and all other bins are normalized to this value.

Testing Procedure (repeat for both versions):

(a) Using a single thread and 100 bins, increase the number of samples until it takes over 100 seconds to run the histogram. This is the size you will use for all further testing

(b) Increase the number of threads from 1 to 256 by powers of 2. Record times taken.

Report: In your report, include a table displaying your timing results for both versions of the code. Graph these times against each other as you increase the number of threads. Write a paragraph explaining the timing results you see, both between the two versions of code, and as number of threads increases.

2. (20 points) Improve the first program by creating a queue for data generation. The threads you create will be split into consumer and producer threads. Producers will generate a random number value, determine the bin in which it is to be placed, and add the bin number to the queue (this is equivalent to both the data generation step and the Whichbin() function). Consumers will grab the bin numbers from the queue as they become available, and place them into the appropriate bin as in part 1. Implementation details will be as follows:

   (a) For simplicity, your queue will consist simply of an array of size 1024. You will have two indices for the queue. One indicates where the next item should be added to the queue. The other indicates where the first that should be consumed is placed. If both indices point to the same location, as they should at the start, that should mean that there are no numbers waiting in the queue. **Note: you will need to account for circling through the end of the array and for potentially filling the array. As a suggestion, a semaphore if initialized properly may be a good way to account for filling the array**

(b) For this portion you should only use the code from part 1 that generate a mutex per bin. Otherwise, there would be little point to having multiple consumers.

(c) For random number generation from each of the producers, you must use the rand_r function for thread safety. I recommend using this Stack overflow link as a reference. If you do not use the thread safe version of random number generation your program may crash without explanation

(d) Aside from the queue and the histogram, there will be no long term data storage. The data variable existing in the original program should not exist.

(e) Since we're now parallelizing both the data creation and the placement of the data into the histogram bins, you should time how long both steps take as a whole.

Testing:

(a) As with part 1, you should first increase the size of your program with 100 bins and a two total threads (in this case you need 1 producer and 1 consumer) until it takes over 100 seconds. Record this time and size and use it for future tests.

(b) Using the size determined above, test for all combinations of 2, 4, 8, and 16 producer and consumer threads (i.e. (2,2), (2,4), ... (4, 2), ... (16, 16)). Show your results in a table in the report. In a couple paragraphs, describe your implementation, and explain why you may be getting the results you do.

(c) (5 points extra credit) Write a setting into your program that prints out an indication when you fill the queue. Using a reasonably small number of consumer threads (8 may be a good start), slowly increase the number of producer threads starting from 1 until you manage to fill the queue. After finding the minimum number of producer threads that will cause the queue to fill, decrease this by 1, and use this ratio of producer and consumer threads for your further testing. Using this ratio of producer to consumer threads, gradually increase problem size and number of total threads until you find the largest problem size you can run in under 100 seconds, as well as the number of producer and consumer threads for doing so. Record the problem size, and number of producer and consumer threads in your report, and in a few sentences discuss why this ratio may be reasonable and how you went about finding the largest problems size possible with this ratio of producer and consumer threads.

Submission Instructions: You will have two submissions. First, your report will consist of a pdf that contains your names, the answers to the first two questions, and the requested tables, graphs, and explanations for the coding problems. Additionally, since we will be testing your code, write exact instructions for how to compile and run your code on csil, as well as a sample large problem size and number of threads to test (aim to run in under a minute). Finally, give a short description of the computer you're running on.

Second, submit your code files together in a zipped folder. You should have 3 separate C files (2 for part 1, one for part 2), a makefile that compiles your programs, and a README that includes your name, your partner's name, and exact instructions to compile and run your three programs.

Turn in Instructions: Please turn in the homework via gradescope by 9 PM on Monday August 1st, 2022. For this assignment, there will be two separate submissions on gradescope. Submit your code in a with an included makefile to hw4_pa and submit your report to hw4. For this programming assignment you will be allowed to submit in pairs. If you are submitting with a partner, please link your partner on your Gradescope submission.