CS5225 Parallel and Concurrent Programming

Take Home Lab 1

Due – June 14 before 11:55 PM

Learning Outcomes

In this lab we will learn how to develop parallel programs using the Pthread library. We will also compare the performance of different implementations. At the end of the lab you will be able to:

- develop a simple program to solve a concurrent problem using threads
- collect performance data and use them to compare the performance of different implementations used to solve a problem

Challenge

Implement a linked list as a:

a) Serial program
b) Parallel program (based on Pthreads) with one mutex for the entire linked list
c) Parallel program (based on Pthreads) with read-write locks for the entire linked list

Your implementation should support *Member*( ), *Insert*( ), and *Delete*( ) functions. Populate the linked list with *n* random, but unique values. Each value should be between 0 and $2^{16} - 1$. Then perform *m* random *Member*, *Insert*, and *Delete* operations on the link list. Let m_{Member}, m_{Insert}, and m_{Delete} be the fractions of operations of each type. You may use any values within 0 and $2^{16} - 1$ while performing these three operations. However, to simplify the implementation, a new value inserted into the list cannot be a value already in the list (it could be a value that was initially added to the list, but later removed).

Step 1: Implement the link list using above 3 approaches.

You may use the code snippets discussed in the class slides as the starting point and complete the missing pieces. [5 marks]

Step 2: Run your program sufficient number of times under the following three cases, and then fill up the given tables using the execution time of your programs. While determining the execution time, consider only the time taken to perform *m* operations and ignore other overheads such as initially populating the link list. Your performance results should be within an accuracy of ±5% and 95% confidence level. See Slides 13-33 and 13-34 by Raj Jain (http://www.cse.wustl.edu/~jain/cse567-08/ftp/k_13cs.pdf) to find out how to calculate the number of samples.

You should run this program on a machine with at least 4 physical cores (Hyper Threading is not considered as a separate core). [3 marks]

Step 3: Plot the average execution time against the number of threads for each case. See http://www.dbooth.net/mhs/common/graphrules.html for some tips on "How to Make a Good Graph". [8 marks]

Step 4: Comment on your observations. Also, list the specification of the machine that you used to run your simulations (see http://dilumb.blogspot.com/2013/12/describing-experimentalsimulation-setup.html for some tips). [4 marks]

Case 1

$n = 1{,}000$ and $m = 10{,}000$, $m_{Member} = 0.99$, $m_{Indert} = 0.005$, $m_{Delete} = 0.005$

Implementation	No of threads					
	1		2		4	
	Average	Std	Average	Std	Average	Std
Serial						
One mutex for entire list						
Read-Write lock						

Case 2

$n = 1{,}000$ and $m = 10{,}000$, $m_{Member} = 0.90$, $m_{Indert} = 0.05$, $m_{Delete} = 0.05$

Implementation	No of threads					
	1		2		4	
	Average	Std	Average	Std	Average	Std
Serial						
One mutex for entire list						
Read-Write lock						

Case 3

$n = 1{,}000$ and $m = 10{,}000$, $m_{Member} = 0.50$, $m_{Indert} = 0.25$, $m_{Delete} = 0.25$

Implementation	No of threads					
	1		2		4	
	Average	Std	Average	Std	Average	Std
Serial						
One mutex for entire list						
Read-Write lock						

Notes

- While the students are expected to find out how to develop programs using Pthreads, students are also encouraged to talk to the lecturer in charge for any clarifications. Discussions on Moodle are encouraged too.

What to Submit

- Submit following files as a single .zip file
  - Source files
  - README.txt explaining how to run your program
- Answers to steps 2, 3, and 4 as a .pdf file
- Name the .zip file as **lab1_<*index no>*.zip**. Replace *<index no>* with your index number