Project #1 - Task Queue Processing with PThreads

SWE3021 Multicore Computing - Fall 2019

Due date: September 30 (Mon) 17:00pm

1 Goal

The primary purpose of this assignment is to gain hands-on experience with multicore parallelism using the standard POSIX Pthreads library. including exposure to multithreading concepts such as mutexes and conditions. The secondary purpose of this assignment is to gain experience implementing a master/worker task queue model of parallel computation.

2 Description

You will develop a parallel version of the serial program you can download from
http://dicl.skku.edu/class/fall2019/multicore/skiplist.h
http://dicl.skku.edu/class/fall2019/multicore/skiplist2.h
http://dicl.skku.edu/class/fall2019/multicore/skiplist.cpp

The program reads a list of "qeuries" from a file. Each task consists of a character code indicating an action and a number. The character code can be either a "i" (for "insert"), "q" (for "qeury"), or "w" (for "wait"). The input file simulates workloads entering a multiprocessing key-value system. In a key-value store system, the "w" actions with number n would insert the number into a skiplist and then update a few global aggregate variables (sum and odd count). The "w" action provides a way to simulate a pause in incoming tasks.

For example, the following script simulates two initial tasks entering the system, followed by a 2-second delay. After the delay, another two insertion tasks and a query task enter the system.

```
i 20
i 15
w 2000
i 10
q 20
i 30
```

Using a purely serial processing system (as implemented in the provided skiplist.cpp), the above scenario will take 8 seconds to finish assuming each task takes 1 second except "i 15" takes 2 seconds:

```
t=0     t=1     t=2     t=3     t=4     t=5     t=6     t=7    t=8
|       |       |       |       |       |       |       |       |
-------------------------------------------------------------------
| i 20  | i 15          | w 2000        | i 10  | q 20  | i 30  | serial version
-------------------------------------------------------------------
```

The final output should match the following, i.e., sorted numbers in the first line, then sum and # odd): If the size of skiplist is greater than 200, printList() function prints only the first 200 elements of the list.

```
10 15 20 30
75 1
```

In this project you will extend this program to take advantage of a multicore CPU using a task queue model. In such a model, the main program spawns a set number of worker threads. You should read the number of worker threads from the command line as a second parameter. The

main program and worker threads communicate using a task queue to keep track of tasks that still need to be processed.

Observe that if we use two worker threads, the same workload can be processed as the following. Note that this is just a example of possible execution scenarios.

```
t=0          t=1           t=2         t=3         t=4
|            |             |           |           |
-------------------------------------------------
| deq,deq    | deq         | deq       | deq, deq  |    master
-------------------------------------------------
| i 20       | w 2000                  | q 20      |    worker 1
-------------------------------------------------
|   i 15                   | i 10      |   i 30    |    worker 2
-------------------------------------------------
```

This can be achieved by splitting the actual processing work out into worker threads that can work parallel to the original master thread. This allows the master thread to focus on receiving jobs while the worker thread focuses on doing the actual work. Note that the the purpose of "w" is to emulate an unexpected delay in worker threads.

Your program should work as follows. At the beginning of execution, the master thread spawns a set number of worker threads (given by a command line parameter). The worker threads are idle at first. Once the workers have been fully initialized, the master then begins to handle tasks from the input file by adding them to a task queue, waking up an idle worker thread (if there are any) for each task. When a thread is awakened, they begin to pull tasks from the queue and process them. If the queue ever runs out of tasks, the worker should block again until awakened by the master. If the worker encounters a "w" (wait) command, it waits the given number of seconds before continuing in the input file. After all tasks have been added to the queue, the master waits for the queue to be exhausted, waking idle threads as necessary to help. When the queue is empty, the master waits for non-idle workers to finish, then sets a global flag to indicate that the entire program is done, re-awakening all worker threads so that they can terminate. The master then cleans everything up and exits.

To implement the above system, you should use Pthread threads, mutexes, r/w locks, or conditions as covered in class. Your program should take the number of worker threads as the second command-line parameter; the performance on parallelizeable workloads should scale linearly with the number of threads.

Note that skiplist2.h is an improved version of skiplist that stores multiple elements in each node. You can use either skiplist.h or skiplist2.h. If you want, you may further improve one of the two implementations. However, note that, if you make changes to skiplist, the critical section region may grow or shrink, which will affect the overall performance when a large number of threads are running.

3 Grading:

1. If you do not submit, you will get 0 points.
2. If your code does not compile, you will get only 1 point.
3. If your code compiles but outputs are incorrect, you will only get 2∼5 points.
4. The remaining 5 points will be the performance score, which is prorated by the following equation.

$$\frac{(the\ slowest\ code's\ execution\ time) - (your\ code's\ execution\ time)}{(the\ slowest\ code's\ execution\ time) - (the\ fastest\ code's\ execution\ time)}$$

4 Class Accounts in In-Ui-Ye-Ji Cluster

In this class, you will have to work on projects on "In-Ui-Ye-Ji" cluster machines. Your ID is your student ID and the initial password is set to your last name in capital, i.e., DOE is your password if your name is John Doe.

SSH command is as follows:

```
$ ssh swin.skku.edu -p 1398 -l your_student_id
```

Note that you have to use ssh port 1398, which is the year when SKKU was first established in Joseon dynasty. The cluster has eight nodes, swin, swui, swye, swji, titan1, titan2, titan3, and titan4. You should be able to login to any node of this cluster using the same id and password.

5 How to Submit

To submit your project, you must run `multicore_submit` command in your project directory in swin.skku.edu server as follows.

```
$ cd your_working_directory
$ multicore_submit project1 ./
```

This command will submit all your files in your current directory to the instructor's account.

For any questions, please post them in Piazza so that we can share your questions and answers with other students. Please feel free to raise any issues and post any questions. Also, if you can answer other students' questions, please don't hesitate to post your answer.