CS 6210: Advanced Operating Systems

Project 1: Thread Scheduling with GTThreads
due: Sunday Feb. 03, 2019 @ 11:59 pm

1. Goal

The goal of the project is to understand and implement credit-based schedulers in a user-level threads library. To do so you must modify the provided GTThreads library. The library implements a O(1) priority scheduler and a priority co-scheduler for reference.

Turn in your final deliverables for the project on **February 3, 2019 by 11:59pm** via Canvas only. Emails or other methods for submission will not be accepted (as explained in more detail below.)

You may discuss ideas with colleagues in the class, but the project must be done individually. Copying others is NEVER allowed for any reason. Please refer to the Georgia Tech honor code available at http://www.honor.gatech.edu/.

2. Project Details

The project minimally requires the following to receive full credit:

- Implement a Credit Scheduler for the provided SMP GTThreads library.
- Provide a way for a user to set the desired scheduler during the GTThreads initialization.
- Implement a function for matrix multiplication, using the provided code. Modify the matrix multiplication files to take an integer as command line argument for type of scheduler (1 for credit scheduler). Set the appropriate value during GTThreads initialization.
- Implement a library function for voluntary preemption (gt_yield()). When a user-level thread executes this function, it should yield the CPU to the scheduler, which then schedules the next thread (per its scheduling scheme). On voluntary preemption, the thread should be charged credits only for the actual CPU cycles used.
- Write a Makefile with the basic rules for compilation and clean up. Also, include a README on how to run your project and other parameters if required.
- Write a final report (in PDF format) summarizing your implementation and detailing the results (as explained below).

Each of these items is explained in the sections below. You may be asked to provide a demo to the TA showing the results.

Directions

Please upload your project on the Canvas website. No emails for submission will be accepted. Use the GTThreads package provided to you as a starting point. The first thing you have to do is to take a look at the provided package very carefully. If you have any questions, you are encouraged to bring them to the TAs, Thaleia Doudali (thdoudali@gatech.edu) and Ke-Jou (Carol) Hsu (nosus_hsu@gatech.edu), during their office hours (to be posted soon) and you can also use the forums on Canvas for posting queries, and to exchange information with other students.

Clone and mirror the project github repo into your private repository:

Before you start coding, create your personal private repo on github and maintain your changes there:

Clone the public project repo:

- git clone https://github.gatech.edu/cs6210-spring19/project1

Mirror to your personal private repo:

- git remote set-url origin https://github.com/<username>/<private-repo-name>.git
- git push -u origin master

3. Multiple Schedulers for GTThreads library

GTThreads library

GTThreads is a user-level thread library. Some of the features of the library are as follows:

- Multi-Processor support: The user-level threads (uthreads) are run on all the processors available on the system.
- Local runqueue: Each CPU has its own runqueue. The user-level threads (uthreads) are assigned to one of these runqueues at the time of creation. Part of the work in the project might involve using some metrics before assigning these uthreads the processor and/or run-time runqueue balancing.
- O(1) priority scheduler and co-scheduler : The library implements these two scheduling algorithms. The code can be used for reference. Priority hash tables used in the library can be used, with some hacking, for the purpose of the credit scheduler. In particular, look at the functions "sched_find_best_uthread_group" and "sched_find_best_uthread".
- You will be able to test your code creating Virtual Machines in an OpenStack cloud. Detailed information about how to set up the testing environment will soon be made available via Canvas. However, it is recommended that you do your initial development and testing on your local machines (most of you must be having at least dual core machines). Once your work is sufficiently stable, you can test it on the virtual machine. We might ask you to demo it on the virtual machines.

4. Credit Scheduler

The credit scheduler is a proportional fair share CPU scheduler built from the ground up to be work conserving on SMP hosts. You would also need to perform migration of uthreads between the kthreads, if a kthread is idle (this maintains the principles of work-conserving and load balancing). Please take a look at the following resources:

- XenWiki for Credit Scheduler (http://wiki.xenproject.org/wiki/CreditScheduler)
- The Xen Credit CPU Scheduler (http://www-archive.xenproject.org/files/summit_3/sched.pdf).
- Comparison of the Three CPU Schedulers in Xen (http://www.xen.org/files/xensummit_4/3schedulers-xen-summit_Cherkosova.pdf).

5. Advice

In order to successfully complete the project, it is recommended that during the first week you are able (i) to understand the GTThreads library, (ii) the mechanisms and algorithm of the credit-based scheduler, and (iii) to have a solid design of the implementation plan you will pursue. It will be helpful to create diagrams illustrating the control flow of the GTThread library.

6. Results and Report

Threading the Matrix Multiplication

A very rudimentary code for multiplying matrices is provided in the matrix directory. The code generates matrices (with each entry as 1) and then creates a number of threads to multiply the matrices. The output of the multiplication corresponds to calculating C = A*B. Each thread calculates a fraction of the rows in C by calculating the same stripe of rows in A times the entire matrix B. Use this code to write a function that multiplies its own matrix. This is to mean that each uthread is working on its own individual matrix.

Test Cases

To get full credit for results, you must execute the following test cases:

- Run 128 uthreads.
- Each uthread will work on a matrix of its own.
- Credits are ranging in {25, 50, 75, 100}.
- Matrix sizes are ranging in {32, 64, 128, 256}.
- So there are 16 possible combinations of credit and matrix size. Since there will be 128 uthreads, there will be 8 uthreads for each combination. (It isn't always interesting to parallelize matrix multiplication. Here multiple sets of threads need to be running over your scheduler with different workloads and priorities).
- Collect the time taken (to the accuracy of micro-seconds) by each uthread, from the time it was created, to the time it completed its task. Also measure the CPU time that each uthread spent running (i.e., excluding the time that it spent waiting to be scheduled) every time it was scheduled.

- The final output should be as follows: For each set of 8 uthreads (having a unique combination of credits and matrix size), print the mean and the standard deviation of both, the individual thread run times, and the total execution times.
- It will be useful if some output is printed while the process is running. For e.g., you may want to print messages when an uthread is put back in the queue, when it is picked from the queue, etc. You can print the output to a per-kthread file instead, if you want.

7. Reporting and summarizing results

To get full credit for the write up, you must include the following:

- Write a report summarizing the implementation of the project.
- Show your understanding of the given GTThreads package (max. 2 pages) - try to jot down how the O(1) scheduler in the package is implemented with simple diagrams (e.g. function interactions or flow chart).
- Present briefly how the credit-based scheduler works (max. 2 pages) - try to explain basic rules (or the algorithm) for scheduling.
- Sketch your design (max. 2 pages) - try to show your implementation plan of the credit-based scheduler involved in the given GTThreads package, that is, how to modify the given package for introducing credit-scheduler into it
- Summarize results for all the test cases above.
- Mention clearly the implementation issues in your final submission. For example, you might want to point out some inefficiencies in your code, etc. This doesn't mean you don't need to fix blatant errors, if there are any minor issues, like performance, don't spend too much time trying to fix it, but make sure you list ways in which you might improve it.

8. Late Policy

A penalty of 5% per day will be applied for late submissions for up to 5 days. Submissions received more than 5 days late will receive no credit.

9. Submission

Please see the submission practice document on how to submit your code. Include all reports, Makefile, etc., in the top directory.