CS 6210: Advanced Operating Systems

Project 1: Buzzing with Threads!
Release Date - Monday, Aug 26, 2019
Due date - Friday, September 20, 2019 11:59 PM

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

Goal

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

Introduction

- **GT Threads Library -** GTThreads is a user-level thread library. Some of the features of the library are as follows:
    a. Multi-Processor support: The user-level threads (uthreads) are run on all the processors available on the system.
    b. Local runqueue: Each CPU has its own runqueue. The user-level threads (uthreads) are assigned to one of these run queues 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.
    c. 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".
    d. You will be able to test your code in class-assigned 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.

- **What is credit scheduler**
    a. 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 the 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:
    b. XenWiki for Credit Scheduler http://wiki.xenproject.org/wiki/CreditScheduler
    c. The Xen Credit CPU Scheduler http://www-archive.xenproject.org/files/summit_3/sched.pdf
    d. Comparison of the Three CPU Schedulers in Xen http://www.xen.org/files/xensummit_4/3schedulers-xen-summit_Cherkosova.pdf

Project Details

---

1. **The project requires you to do the following to receive full credit:**
    a. Implement a Credit Scheduler for the provided SMP GTThreads library.
    b. Provide a way for a user to set the desired scheduler during the GTThreads initialization.
    c. Implement a function for matrix multiplication, using the provided code.
        i. For this, you are required to modify the matrix multiplication files in a way that it takes an integer as a command line argument for the type of scheduler (0 for priority scheduler, 1 for credit scheduler) and set the appropriate value during GTThreads initialization.
    d. 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.
        i. Have a command line argument that will enable/disable the yield functionality.
        ii. For this, you are required to implement a library function for voluntary preemption gt_yield()
    e. Load balancing - Implement uthread migration if a kthread is idle.
        i. Have a command line argument that will enable/disable load balancing.
    f. Code execution -
        i. Write a Makefile with the basic rules for compilation and clean up.
        ii. Include a README on how to run your project and other parameters if required.

g. Write a final report (in PDF format) summarize your implementation and detailing the results.

- **Deliverable Phases**
  a. The above list of requirements is divided into phases to make sure you are able to build the project in a timely manner. We recommend you follow the phases as it will help you achieve the goal of the project.
     i. **Phase 1 -** Understand the given skeleton code. Once you have an idea about the flow go ahead and create flow diagrams illustrating
        - High level control flow for scheduling in the GTThread library.
        - Flow chart for O(1) priority scheduler.
     ii. **Phase 2** : Now since you have the understanding of the library flow it's time to do some implementation
        - Have a flow chart for the design.
        - How will you verify your design - think and discuss on the piazza on how to test credit scheduling is working or not?
        - Start writing the code.
     iii. Mid-check-point (see deadlines for the date)
     iv. **Phase 3** : Add Yield and Load Balancing functionalities.
     v. **Phase 4** : Start compiling the report and make sure you don't miss
        - The flow diagrams.
        - Design implementation.
        - How Yield and Load balancing is implemented and their impact
        - Results.

        Detailed information about the report can be found below in the Report section.

- **Location** - https://github.gatech.edu/cs6210-fall2019/project_1

Evaluation

Setup

1. You are required to run 128 uthreads. Each uthread will work on one matrix by itself. For a given uthread you have the following options
   - Credits equals {25, 50, 75, 100}.
   - Matrix sizes (M) equals - {32, 64, 128, 256}.

2. So there are 16 (4 X 4) possible combinations of credit and matrix size (also termed as group_name), for example - credit = 25 and matrix = 32 (c_25_m_32) or credit = 50 and matrix = 32 (c_50_m_32).

3. Every combination of credit and matrix is executed by 8 threads. *Each thread multiplies two (M,M) matrices independent of other threads.* (16 combinations, each having 8 threads totals to 128 uthreads.)

Testing

- Terminologies
    a. CPU Time - time spent by a uthread running every time it was scheduled (i.e. excluding the time it spent waiting to be scheduled).
    b. Wait Time - time spent by a uthread waiting in the queue, to be scheduled.
    c. Execution Time - CPU Time + Wait Time

1. **Testing Credit Scheduler -** is tested <u>without</u> load balancing enabled. We have provided two files in the output directory.

    a. Detailed_output.csv - records the per uthread statistics.

group_name	thread_number	cpu_time(us)	wait_time(us)	exec_time(us)
c_25_m_32	0			
c_25_m_32	1			
c_25_m_32	2			

    b. Cummulative_output.csv - capture the aggregate statistics of uthreads belonging to the same group_name.

group_name	mean_cpu_time	mean_wait_time	mean_exec_time	sd_cpu_time	sd_wait_time	sd_exec_time
c_25_m_32						
c_25_m_64						

    c. Graphs: Plot a split bar graph showing mean_exec_time (composed of mean_cpu_time (bottom) and mean_wait_time (top)) vs group_name, keeping the matrix size constant while varying the credits {25, 50, 75,100}. A separate plot is expected for every matrix size.

    - **The file should be named m_<i>.png** (where i == fixed matrix size)

2. **Testing Yield -**
   Measure the CPU cycles from the beginning of the time slice to yield. Display the amount of credits deducted and remaining after the yield. The reduction should be proportional to the amount of CPU time spent. Also, print queues of the kthread pre and post yield. Have a flag that can enable / disable this functionality.

   In the report, explain the mechanism used to implement yield and also include screenshots of the queues, while during the demo it will be tested using the command line argument.

3. **Load Balance -** Print queues of kthreads post and pre for every migrating uthread. This should show the uthread migrated from one kthread to another. Also, plot split bar graphs as mentioned earlier to show the total execution time.

   In the report, explain the mechanism used to implement load balancing and also include screenshots of the queues, while during the demo it will be tested using the command line argument.

   The split bar graph file should be named lb_m_<i>.png (where i == fixed matrix size)

Output

Important Note - Please make sure the .csv files have the same name as provided, with columns in the same order and are placed in the output/csv directory. Similarly, .png graph files should have the naming convention as mentioned above and should be placed in output/graphs directory. ***The evaluation scripts will only look for the given name/format and any deviation will not be evaluated.***

Resources

- We will provide you the details of Virtual Machines that you will be using to test the code. You are required to use the VM to get the evaluation results through the Virtual Machines provided by us. The VM will be used during the demo session as well. No personal/other machines will be allowed.
  - **How to access VM - <to be updated>**
- Private repo creation - We will highly recommend to use private repos to regularly backup your code. **<to be updated>**
- We have spent time to develop FAQs for the project do check them.

Report

Report Template – **to be shared soon! (Before the class on Thursday)**

Evaluation - Scoring Matrix

Results	**Report**	**Demo**	**Total**
30	40	30	100

Fine grain scoring matrix -

- Results

Task	**Description**	**Total = 30 points**
Detailed Output File		10
Cumulative Output File		10
Graphs		10

- Report

Task	**Description**	**Total = 40 points**
Goal of the project	Explain in your words the goal of the project	5 points
Control Flow Diagram		10 points
Design Description		15 points
Evaluation and Results		10 points

- Demo

Task	**Description**	**Total = 30 points**

Code & Questions	The code will be downloaded from the submission and is executed. Questions will be asked related to the code/project.	30 points

Submission Instructions

Please see the submission practice document in the project directory.

Deadlines

- **Release Date** : Monday, Aug 26, 2019
- **Mid-check-point** : This is a guideline aimed to have you figure out at what stage of the project completion you are at. Make sure you are done with Phase 1 and 2 by September 12, Thursday.
- **Due date - Friday, September 20, 2019 11:59PM - Final report and code**

Communication Policy

Our effort is to ensure we can address the problems of all students in a prompt manner and students help each other (while honoring the Georgia Tech honor code as mentioned above). Additionally, we would like to make sure everyone is aware of the solutions to general doubts and problems faced as you work on your projects. Thus, to have transparent communication we are following the below given policy -

1. All the questions related to project/class must be asked on the Piazza.
2. Emails to the TAs should be done in cases such as submission issues or unless asked due to an issue affecting an individual only.

If an email is sent -

1. If an email is sent - make sure the email is sent to both the TA's
    a. Ranjan Sarpangala Venkatesh | ranjansv@gatech.edu
    b. Harshit Daga | harshitdaga@gatech.edu
2. To ensure emails are answered in a timely manner we have our filters set for the subject pattern [CS-6210]. Please make sure to add [CS-6210] before your subject.
   For example [CS-6210] Is George P. Burdell real?

Late Submission

---

1. A flat penalty of 30% for late submission will give you an extension by 3 days.
2. To avail late submission - email must be sent to the TAs within 12 hours of the deadline.
3. TAs will let you know about the late submission process.