Name: HyunJong Lee

GTID: 902987836

Email: [hlee652@gatech.edu](mailto:hlee652@gatech.edu)

Class: CS4210 - A

Final Report for GTThreads

**GTThreads package:**

O(1) scheduler:

GT thread package implements O(1) scheduler for insertion, removal and retrieval of next uthread to be scheduled from the queue. For each of kthread, which acts like a virtual CPU, has two runqueues: active and expired respectively. When runqueue is initialized, it puts all the tasks into active runqueue. When each task is preempted, it puts into expired runqueue. Once the active runqueue becomes empty, it swaps the active and expired runqueue. Swapping two runqueues is done by swapping two pointers and can be done in O(1) time.

<DIAGRAM>

**Credit-based scheduler:**

How it works:

Credit scheduler works somewhat similar way as priority queue scheduler works. At instantiation state, it assigns default credit to each of uthreads. Given the credit, each uthread runs for given amount of time , calculated based on credit in milliseconds. Once the uthread is scheduled, it loses certain amount of credit, based on time it sent on CPU. Until the credit remains in positive number, it resides in active runqueue. However, as soon as it consumes all the credit, it is moved to expired runqueue with newly assigned credit (default credit).

Implementation in GTThreads:

Each of uthread is assigned default credit and credit left in uthread\_create() and added to active runqueue:

u\_new->credits.credit\_left = u\_new->credits.def\_credit = credits;

add\_to\_runqueue(kthread\_runq->active\_runq, &(kthread\_runq->kthread\_runqlock), u\_new);

When VTALRM signal is received by one of the kthreads, the handler function ksched\_priority() is invoked to schedule a new uthread by calling uthread\_schedule() function.

uthread\_schedule() function is where credit scheduler algorithm is implemented. Within the function, it calls calculate(), which updates the time it is updated and credit left, which is decreased by current time – last updated time in millisecond.

If the credit is less than 0, it adds to expired runqueue. Otherwise, it adds to active runqueue. Before calling the setlongjmp(), it sets the timer. If the uthread is first time scheduled (== def\_credit), it uses kthread\_init\_vtalrm\_timeslice() to set timer. Otherwise, it converts remaining credits into millisecond and set the timer.

For the load balancing part, it briefly checks whether it’s first time scheduled thread. If so, it goes through kthreads array and add into specific runqueue.

To cover gt\_yield() API call, I added variable yid, to check voluntarily preempted case. Note that, the real work (computation), is done in uthread\_context\_func().

**Summarization of results for all the test cases:**  
The following chart shows average and standard deviation for both running time (on CPU) and total execution time for each of unique pair (metrics size and credit).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metrics Credit | Avg. Running time | Avg. Execution time | Standard deviation Running time | Standard deviation Execution time |
| ( 32x 32 25) | 19 | 1107 | 2 | 121 |
| ( 32x 32 50) | 17 | 1080 | 1 | 122 |
| ( 32x 32 75) | 17 | 827 | 1 | 451 |
| ( 32x 32 100) | 17 | 237 | 1 | 427 |
| ( 64x 64 25) | 133 | 1563 | 13 | 127 |
| ( 64x 64 50) | 130 | 1427 | 11 | 126 |
| ( 64x 64 75) | 131 | 1391 | 9 | 88 |
| ( 64x 64 100) | 134 | 1252 | 2 | 77 |
| (128x128 25) | 1043 | 4801 | 845 | 1313 |
| (128x128 50) | 1024 | 4191 | 103 | 909 |
| (128x128 75) | 1027 | 3921 | 1409 | 520 |
| (128x128 100) | 1050 | 3027 | 77 | 839 |
| (256x256 25) | 9835 | 32744 | 1880 | 3018 |
| (256x256 50) | 8915 | 24977 | 362 | 2856 |
| (256x256 75) | 8979 | 24907 | 130 | 2973 |
| (256x256 100) | 8682 | 15985 | 771 | 2789 |

Regardless of amount of credit given, it shows similar running time on CPU for same metrics size. That is to say, even though the thread with higher credit finishes the job early, at the end, it takes same amount of time in CPU on doing its job. A notable difference is average total execution time. As we expected, the higher credit it has, the lower waiting time it takes. Since a thread with higher credit gets more timeslice in given order, it can reside in CPU for longer time and finish the job early. Therefore, it waits less to load itself onto the CPU.

**Implementation Issues:**

Despite the uthread in general is implemented sufficiently, it outputs illegeal instruction or segmentation fault frequently. In such case, you can run multiple times until you get the statistics.