GTThreads with Credit Scheduler

Jin Wang
903009287

I. Introduction

The goal of the project is to understand and implement credit-based schedulers in a user-level threads library. The library implements a O(1) priority scheduler and a priority coF scheduler for reference.

II. Priority scheduler and co-scheduler

In the given package, priority scheduler and co-scheduler are implemented.
gt_kthread.c: create and execute kthreads.
gt_uthread.c: create, execute uthreads and schedule uthreads according to their value of priority.
gt_pq.c: add/remove uthreads of a kthread runqueue.
gt_signal.c: VTALRM used as timer interrupt to kthreads; SIGUSR1 relays the VTALRM signal all other kthreads; SIGUSR2 run uthreads in a separate context and stack.
gt_spinlock.c: interface for the synchronization of shared resources.

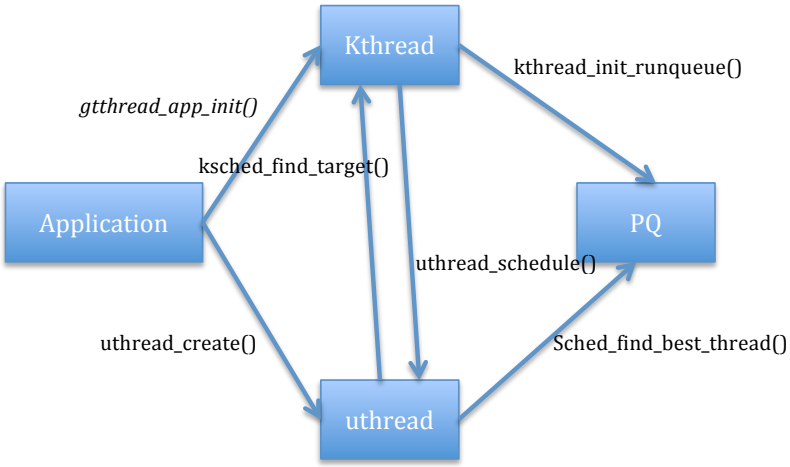

Figure1: Important inter-module interaction in GTThread

III. Credit scheduler Implementation

1. Credits and Timeslice

Every kthread has a runqueue of uthreads. Every uthread is assigned with a credit value when created: `uthread_struct_t->credits`. Uthread burns 25 credits every 10ms (`CREDITS_BURN_TSLICE`). Its priority will also be recalculated.
If a uthread is still in UNDER priority after 30ms(`YIELD_TSLICE`) of running time, it will be preempted by call `uthread_schedule()`.
The credits of OVER uthreads will not be added again.

2. Priority

There are only two types of priority in Credit scheduler : UNDER(`#define UTHREAD_PRI_UNDER 0`) when credits > 0 and OVER(`#define UTHREAD_PRI_OVER 1`) when credits<=0. Kthread schedules all the in queue UNDER uthread by FIFO. Thus, `uthread_schedule(&sched_find_best_uthread)` is still used in this implementation.

3. Time collection

To collect the life time and running time of each uthread, three time values are defined for time collection with function of `clock_gettime(): start_time, end_time and create_time`.

```
extern int uthread_create(){
clock_gettime(&u_new->create_time);
}
extern void uthread_schedule(){
if((u_obj->uthread_state & (UTHREAD_DONE | UTHREAD_CANCELLED))){
clock_gettime(&u_obj->end_time);
u_obj->running_time += (u_obj->end_time - u_obj->start_time) ;
u_obj->life_time += (u_obj->end_time - u_obj->create_time) ;
}else {
clock_gettime(&u_obj->end_time);
u_obj->running_time += (u_obj->end_time - u_obj->start_time) ;
}
u_obj->uthread_state = UTHREAD_RUNNING;
clock_gettime(&u_obj->start_time);
}
```

IV. Sched_yield() implementation

By calling sched_yield(), A process can relinquish the processor voluntarily without blocking. The process will then be moved to the end of the queue for its static priority and a new process gets to run.
In GTThreads, sched_yield() function is implemented in the matrix structure. When a running uthread calls `yield()`, it stops voluntarily and system schedules another runnable uthread with `uthread_schedule(&sched_find_best_uthread)`.

To test this function, uncomment `#define SCHED_YIELD` in gt_matrix.c.

V. Result

In this project, 128 uthreads of matrix multiplication are divided by 16 groups according to their size of matrix and initial credits. 8 uthreads in each group.

size \ credits	25	50	75	100
32	0	1	2	3
64	4	5	6	7
128	8	9	10	11
256	12	13	14	15

Tab1. Group number of 16 matrix_size/credits combinations

Output of the mean and the stander deviation of `life_time` and `running_time` can be found in output.log.

1. Mean of life time

size \ credits	25	50	75	100
32	176,062.625	176,275.750	176,577.875	176,767.250
64	178,655.500	180,497.375	182,151.250	184,154.250
128	199,079.750	214,349.625	335,492.500	243,582.000
256	1,588,745.500	1,534,066.125	1,352,179.500	1,369,275.750

Tab2. Mean life time of 16 groups of uthreads.

In the process of implement, it's observed that only uthreads with matrix size of 256 executed the `burn_credits()` function. Thus the results of these 4 groups more persuasive.

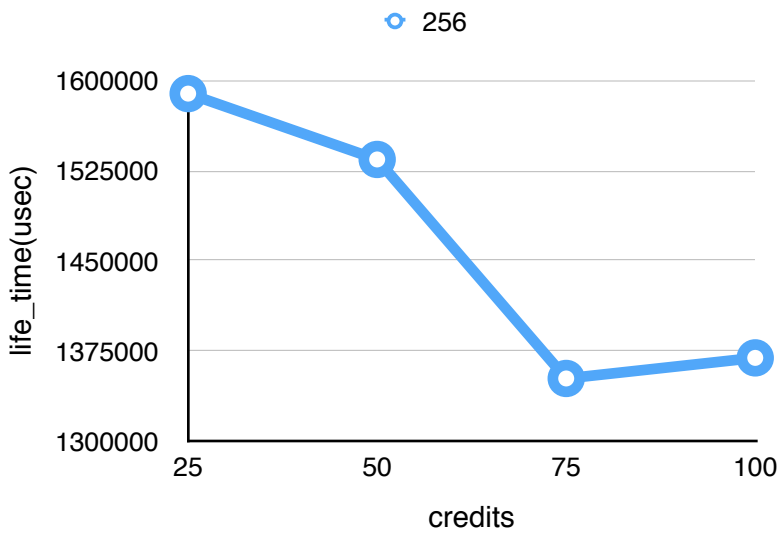

Figure 2. Mean life time of 256 matrix_size uthreads as a function of credits.

In Figure 2, the mean running time of uthreads decreases with the greater value of credits. It can be explained as: higher credits uthreads have more proportion of time of being in UNDER priority; lower credits uthreads take more time of waiting in queue with a OVER priority.

2. Mean of running time

size \ credits	25	50	75	100
32	230.875	209.750	299.000	184.750
64	1,885.250	1,838.875	1,562.750	1,998.625
128	14,919.250	15,264.875	14,989.625	14,981.875
256	174,858.375	200,176.875	191,653.125	177,249.250

Tab3. Mean running time of 16 groups of uthreads.

On the contrary, the evaluation of mean running time is not observable within same size of matrix, but they increase with size of matrix increasing.

VI. Reference

1. GTThread:A User-level thread Library, Dipanjan Sengupta
2. https://github.com/suokun/xen_credit_scheduler.c/blob/master/xen_credit_scheduler.c
3. http://wiki.xen.org/wiki/Credit_Scheduler