GTThread: A User-level thread Library

Dipanjan Sengupta

1. Modules

There are majorly 5 modules in GTThread lib:

1. gt_kthread.c: Handles the creation and execution of *kthreads* that simulates virtual processors. The user-level threads (uthreads) created by the application are scheduled on kthreads.
2. gt_uthread.c: Handles the creation, scheduling and execution of user-level threads (uthreads).
3. gt_ pq.c: Priority runqueue. Handles the O(1) time insertion, removal and retrieval of next uthread to be scheduled from the queue. There are two runqueues associated with each virtual CPU (kthread), *active* and *expired* queue. Tasks are scheduled from active queue and preempted tasks are inserted back into the expired queue. If the active runqueue is empty then active and expired queues are swapped, and then scheduling proceeds as usual.
4. gt_signal.c: Handles the signaling used for scheduling. VTALRM signal is used as the timer interrupt to the virtual CPUs (kthreads). SIGUSR1 signal is used to relay the VTALRM signal to all other kthreads. SIGUSR2 signal is used to run the uthread in a separate context and stack.
5. gt_spinlock.c: Provides the interface for the synchronization of shared resources.

2. Application

For the application to use the thread library it has to make 3 calls in the following sequence:
gtthrad_app_init(): Initialize the library
uthread_create(): uthread creation.
Gthread_app_exit(): Exit from the library

3. kthread module

Following is the control flow in kthread module when application calls gtthread_app_init() function:
gtthread_app_init() -> kthread_create() -> kthread_handler() -> kthread_init().

In kthread_init(): Timer and relay signals are registered, and runqueues are also initialized.
When VTALRM signal is received by one of the kthreads, the handler *ksched_priority()* is invoked. *ksched_priority()* then relays this signal to all other kthreads. So the basic action of both the handlers is to schedule a new uthread by calling uthread module function uthread_schedule().
ksched_priority()->uthread_schedule()
ksched_cosched()->uthread_schedule()

4. uthread module

Following is the important call stack (stack is compressed because there are signal passing involved) in uthread module:
Uthread_schedule() -> uthread_init() -> kill(SIGUSR2) -> uthread_context_func()

Uthread_init() raises a SIGUSR2 signal to itself so that the control goes to the signal handler *uthread_context_func()*. *uthread_context_func()* is the function where a scheduled uthread actually starts its execution. The execution context is saved and resumed using sigsetjmp() and siglongjmp() calls.

5. Priority Queue module

This module contains the data structure to implement the runqueues. There are two runqueues per virtual CPU (kthread), active and expired queue. Basic functions like *init_runqueue()*, *add_to_runqueue()*, *remove_from_runqueue()* and *switch_runqueue()* are well defined and does the same job as their names suggest.

The function that finds the highest priority uthread from the active runqueue is *sched_find_best_thread().* Following is the brief description of the function:

1. The lowest set bit in the uthread_mask gives the highest priority index.
2. Using the priority index and the priority array (prio_array) in the runqueue data structure, corresponding prio_structure_t element is accessed.
3. The lowest set bit in the group_mask in the priority element gives the uthread group id.
4. Using the uthread group id the first uthread element in the thread group is picked by the scheduler to be scheduled to the corresponding kthread.

We can see that uthread selection is done in O(1) time. During preemption, corresponding uthread is pushed to the tail of the expired queue. So the insertion operation also happens in constant time.

6. Module interaction

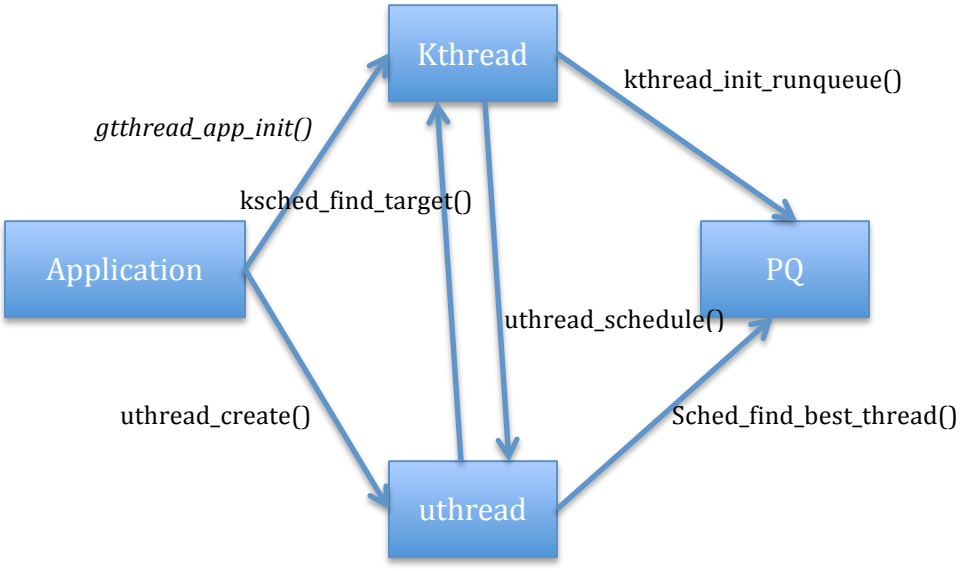

Figure1: Important inter-module interaction in GTThread