An Implementation of Round-Robin Scheduling for RIOT OS

Thanaphon Leonardi
University of Genova, DIBRIS
Via All'Opera Pia, 13
S4633433@studenti.unige.it

Dylan Lo Blundo
University of Genova, DIBRIS
Via All’Opera Pia, 13
S4623731@studenti.unige.it

Summary

A Round-Robin scheduling policy has been implemented with a time quantum of 0.5 seconds, replacing the previous priority-based approach.

Priorities have been maintained, however, in the thread structure, to maintain as much of the original operating system’s structure as possible.

The RRTester application has been developed to verify the correct functionality of the new scheduler.

1 Introduction

A new chronologically ordered list has been created to keep track of all threads, including the idle thread (for which specific checks are made).

Due to the requirements of the **RRTester** application, the thread structure has been expanded to also include information on the service times of the various threads. To this end, the **xtimer** library has also been used, allowing the calculation of remaining service times and the switching of threads every quantum of time.

Finally, the *runqueue bitcache* and its relative system checks have been removed, to reduce superfluous work load.

2 Implementation Choices

2.1 File Changes

- *sched.c*
- *sched.h*
- *thread.c*
- *thread.h*

2.2 Tester Application

main.c

The **RRTester** application creates five threads, each with its own name and service time. To simulate the required workloads, the threads are put into busy waiting until the specified time has passed.

2.3 Thread Structure

thread.h

struct _thread

```c
struct _thread {
    char *sp;                         /**< thread's stack pointer         */
    thread_status_t status;           /**< thread's status                */
    uint8_t priority;                 /**< thread's priority              */
    uint64_t service_time;            /**< thread's total executed time   */
    uint64_t max_service_time;        /**< thread's maximum lifetime      */

    kernel_pid_t pid;                 /**< thread's process id            */
```

service_time and max_service_time have been added to thread structure.

2.4 Thread Implementation

thread.c

thread_create()

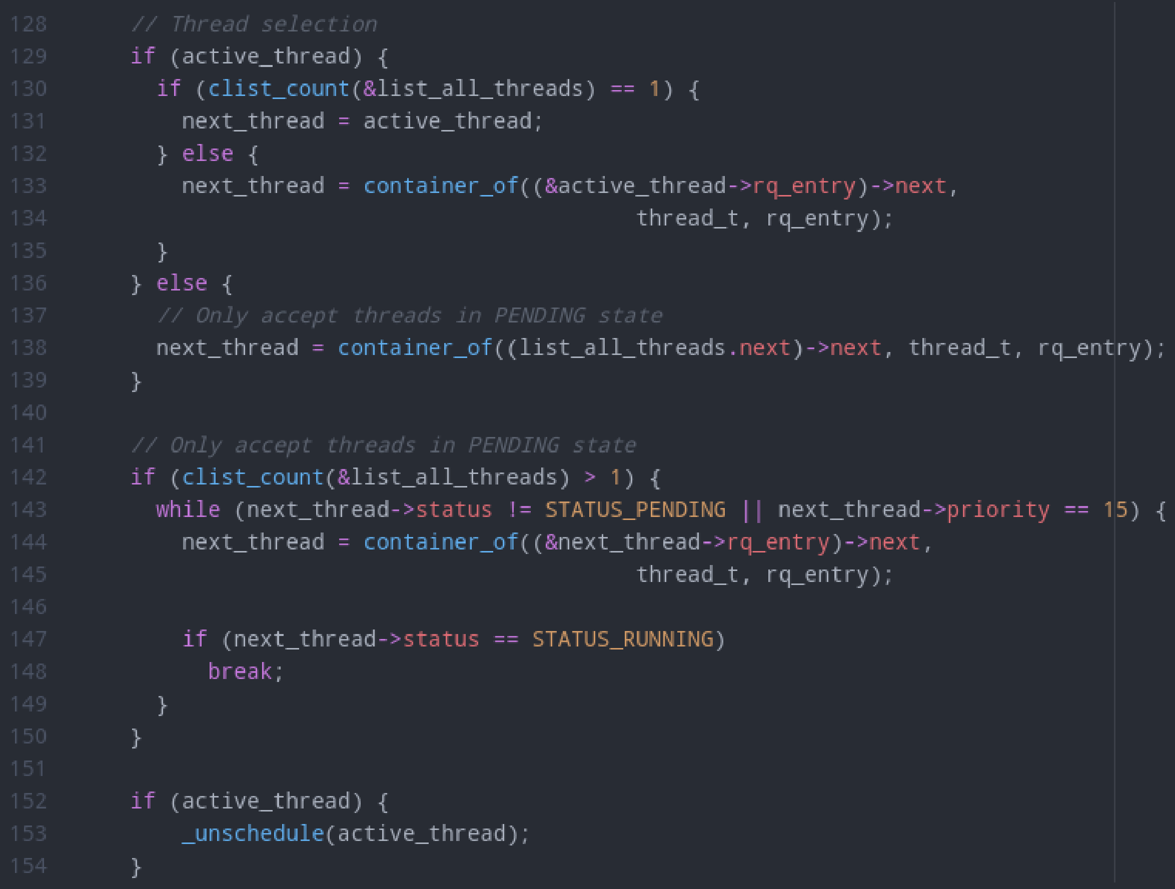
The necessary changes have been made to keep the new thread list (list_all_threads) updated.

The idle thread, which behaves differently, is ignored by the Round-Robin queue until it is the last remaining one.

2.5 Scheduling Algorithm

sched.c

sched_run()



```
  // Thread selection
  if (active_thread) {
    if (clist_count(&list_all_threads) == 1) {
      next_thread = active_thread;
    } else {
      next_thread = container_of((&active_thread->rq_entry)->next,
                                       thread_t, rq_entry);
    }
  } else {
    // Only accept threads in PENDING state
    next_thread = container_of((list_all_threads.next)->next, thread_t, rq_entry);
  }

  // Only accept threads in PENDING state
  if (clist_count(&list_all_threads) > 1) {
    while (next_thread->status != STATUS_PENDING || next_thread->priority == 15) {
      next_thread = container_of((&next_thread->rq_entry)->next,
                                       thread_t, rq_entry);

      if (next_thread->status == STATUS_RUNNING)
        break;
    }
  }

  if (active_thread) {
      _unschedule(active_thread);
  }
```

The thread selection algorithm scans through the queue until it finds the next *pending* thread (ready for execution). Appropriate checks have been implemented to ignore the idle thread, should there be other threads in the list.

2.6 Service Time Calculation

sched.c

sched_run()

```c
    if (service_time_start != 0) {
      next_thread->service_time += xtimer_now64().ticks64 - service_time_start;
    }

    if (next_thread->priority != 7) {  // if NOT main thread
      service_time_start = xtimer_now64().ticks64;
    }

    sched_active_pid = next_thread->pid;
    sched_active_thread = (volatile thread_t *)next_thread;

    #ifdef DEVELHELP
      if (!isIdle) {
        printf("\nCurrently running thread: Thread %s\n", sched_active_thread->name);
      }

      if (next_thread->priority == 15) {
        isIdle = true;
      } else {
        isIdle = false;
      }
    #endif

    if(next_thread->max_service_time != 0) {
      if (next_thread->max_service_time > next_thread->service_time) {
        // Avoid overflow from subtracting unsigned numbers
        float time = (float) ((next_thread->max_service_time - next_thread->service_time));
        printf("Thread service time remaining: %.2fs\n", (float) time / 1000000);
      } else {
        printf("Thread service time remaining: 0.00s\n");
      }
    }

```

In this section, the calculation of the remaining service time; service_time_start represents the moment in which the last thread woke up, which is then subtracted from the current time (timer_now64().ticks64), returning the remaining time.

The first if ensures no calculation occurs when an initial time has not yet been assigned, while the second if disallows the main thread from being included in the calculation.

```c
    xtimer_set(&timer_rr, ROUND_ROBIN_TIME_QUANTUM);
```

This is the callback function: every time quantum, it executes the scheduling algorithm.