Corso di Sistemi Operativi a.a. 2022-23

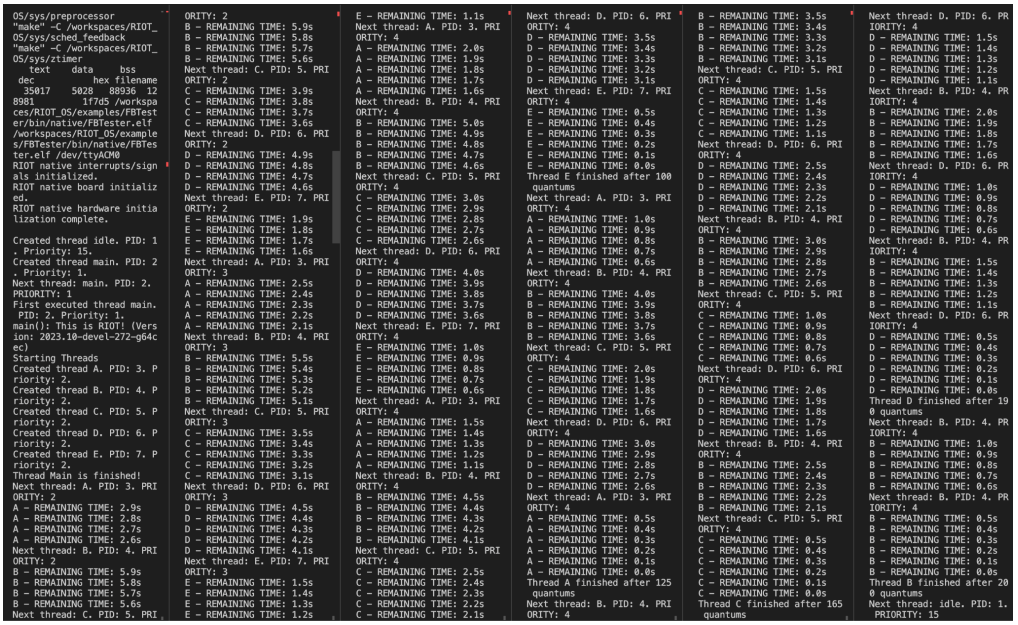
# FEEDBACK SCHEDULER IN RIOT OS

***Trebino Nicolò, Garau Federico***
DIBRIS, Università di Genova

## SOLUTION

We decided to implement a feedback scheduler that replaces the original priority-based scheduler. The feedback scheduler that we implemented makes all the threads created with the “thread_create” function with a fixed priority of 2, the only exceptions are for the Main and the Idle threads. The Main has the highest priority (1) and the Idle has the lowest (15), and the queue of their priority works as a FIFO algorithm (notice that it has no sense to create a thread with priority 15 because it won’t be executed because of the Idle thread is an infinite loop), all the other threads will start with a priority of 2 in the queue of priority 2, and after the time quantum, set to 0,5s, they will be moved into the queue of priority 3, and with the “thread_yield_higher()” the thread will yield only if there is another thread with a higher priority (lower number is a higher priority). After reaching the third and last queue of our feedback scheduler (so the queue with priority 4), if there is no higher priority thread, the scheduler will use a Round-Robin algorithm between the threads in the queue with priority 4.
For testing, we have created the main.c, located in the:
/RIOT_OS/examples/FBTester/main.c.
We created, as requested, 5 different threads named A, B, C, D, and E with different service times.
With the defined global constant QUANTUM (milliseconds) you can set the time interval between every print of the specified thread.
Here you have some output examples with different values of QUANTUM.

QUANTUM 100



QUANTUM 500

```
RIOT native interrupts/signals initialized.
RIOT native board initialized.
RIOT native hardware initialization complete.

Created thread idle. PID: 1. Priority: 15.
Created thread main. PID: 2. Priority: 1.
Next thread: main. PID: 2. PRIORITY: 1
First executed thread main. PID: 2. Priority: 1.
main(): This is RIOT! (Version: 2023.10-devel-275-gf1e278)
Starting Threads
Created thread A. PID: 3. Priority: 2.
Created thread B. PID: 4. Priority: 2.
Created thread C. PID: 5. Priority: 2.
Created thread D. PID: 6. Priority: 2.
Created thread E. PID: 7. Priority: 2.
Thread Main is finished!
Next thread: A. PID: 3. PRIORITY: 2
Next thread: B. PID: 4. PRIORITY: 2
Next thread: C. PID: 5. PRIORITY: 2
Next thread: D. PID: 6. PRIORITY: 2
Next thread: E. PID: 7. PRIORITY: 2
Next thread: A. PID: 3. PRIORITY: 3
A - REMAINING TIME: 2.5s
Next thread: B. PID: 4. PRIORITY: 3
B - REMAINING TIME: 5.5s
Next thread: C. PID: 5. PRIORITY: 3
C - REMAINING TIME: 3.5s
Next thread: D. PID: 6. PRIORITY: 3
D - REMAINING TIME: 4.5s
Next thread: E. PID: 7. PRIORITY: 3
E - REMAINING TIME: 1.5s
Next thread: A. PID: 3. PRIORITY: 4
A - REMAINING TIME: 2.0s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 5.0s
Next thread: C. PID: 5. PRIORITY: 4
C - REMAINING TIME: 3.0s
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 4.0s
Next thread: E. PID: 7. PRIORITY: 4
E - REMAINING TIME: 1.0s
Next thread: A. PID: 3. PRIORITY: 4
A - REMAINING TIME: 1.5s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 4.5s
Next thread: C. PID: 5. PRIORITY: 4
C - REMAINING TIME: 2.5s
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 3.5s
Next thread: E. PID: 7. PRIORITY: 4
E - REMAINING TIME: 0.5s
E - REMAINING TIME: 0.0s
Thread E finished after 20 quantums
Next thread: A. PID: 3. PRIORITY: 4
A - REMAINING TIME: 1.0s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 4.0s
Next thread: C. PID: 5. PRIORITY: 4
C - REMAINING TIME: 2.0s
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 3.0s
Next thread: A. PID: 3. PRIORITY: 4
A - REMAINING TIME: 0.5s
A - REMAINING TIME: 0.0s
Thread A finished after 25 quantums
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 3.5s
Next thread: C. PID: 5. PRIORITY: 4
C - REMAINING TIME: 1.5s
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 2.5s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 3.0s
Next thread: C. PID: 5. PRIORITY: 4
C - REMAINING TIME: 1.0s
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 2.0s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 2.5s
Next thread: C. PID: 5. PRIORITY: 4
C - REMAINING TIME: 0.5s
C - REMAINING TIME: 0.0s
Thread C finished after 33 quantums
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 1.5s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 2.0s
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 1.0s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 1.5s
Next thread: D. PID: 6. PRIORITY: 4
D - REMAINING TIME: 0.5s
D - REMAINING TIME: 0.0s
Thread D finished after 38 quantums
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 1.0s
Next thread: B. PID: 4. PRIORITY: 4
B - REMAINING TIME: 0.5s
B - REMAINING TIME: 0.0s
Thread B finished after 40 quantums
Next thread: idle. PID: 1. PRIORITY: 15
```

## IMPLEMENTATION CHOICES

We wanted to keep the RIOT OS structure as clean as possible and make our changes only to be a module that you can use or not use, we achieved this using functions already implemented in the operating system, like sched_change_priority() and/or the “#if ” directive to enable some modification only if the module “MODULE_SCHED_FEEDBACK” is used.
We added only two important files in the RIOT source code: the sched_feedback.h and the sched_feedback.c, both located in the sys folder (obviously we have also created all the Makefiles to be able to compile and run our application correctly through our new scheduler).
In the sched_feedback.h we have defined only the timer that we used for the new scheduler and the sched_feedback_init, the function used to initialize the feedback scheduler.
In the sched_feedback.c we have implemented the sched_feedback_init and we implemented also three important and different functions:

- sched_feedback_set → which sets the feedback scheduler if it’s necessary
- sched_runq_callback → which checks if the priority of the active thread is the right priority and it decides what to do based on the priority of the active thread passed as a parameter.
- sched_feedback_cb → this function is called every tick of the timer (we used the ztimer to implement it) and it decides what to do based on the priority of the active thread.

Some parts of the code are commented for a better understanding of the behavior of the scheduler.

## INSTRUCTION FOR BUILDING, EXECUTING, AND TESTING

Building our application is easy.
A Linux machine is required.
Prerequisite:

- Essential system development tools (GNU Make GCC, standard C library headers)
- git
- GDB in the multiarch variant (alternatively: install for each architecture you target the corresponding GDB package)
- unzip or p7zip
- wget or curl
- python3
- pyserial (linux distro package often named python3-serial or py3-serial)

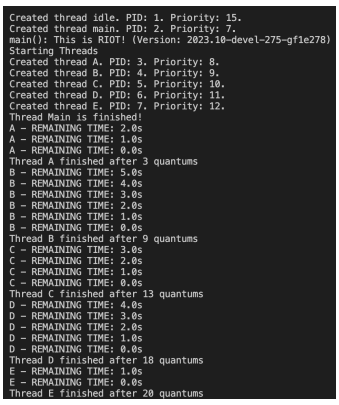
In Ubuntu for example you can install all these packages with this command:
“sudo apt install git gcc-arm-none-eabi make gcc-multilib libstdc++-arm-none-eabi-newlib openocd gdb-multiarch wget unzip python3-serial doxygen”

If you are reading this document I suppose that you have already unzipped our folder, so go into this folder directory on the terminal and clone the RIOT repository using this command:
“git clone https://github.com/RIOT-OS/RIOT.git” #this is to clone the RIOT repository

Now apply our changes with this command:
“git apply RIOT_patch.patch” #this to apply the patch file

To test the application you have to go into the directory “RIOT/examples/FBTester” and run the application with these commands:
“cd RIOT/examples/FBTester” #for entering in the directory of the application

“make all term” #for compiling and running the application




To test the differences with the standard scheduler you can also try to use the standard scheduler already implemented in RIOT with this command instead of the last one:
“NOFB=1 make all term”.
Here on the left side, you can find the output:
NOFB = 1 with QUANTUM set to 1000