Real Time Operating Systems Design and Programming

Lab 4

OS Scheduler Lab

Evaluating Scheduler Responsiveness

Contents

1 Overview

In this lab you will evaluate how changing the scheduling approach (prioritisation and pre-emption) affects how quickly a multithreaded program responds to an input event.

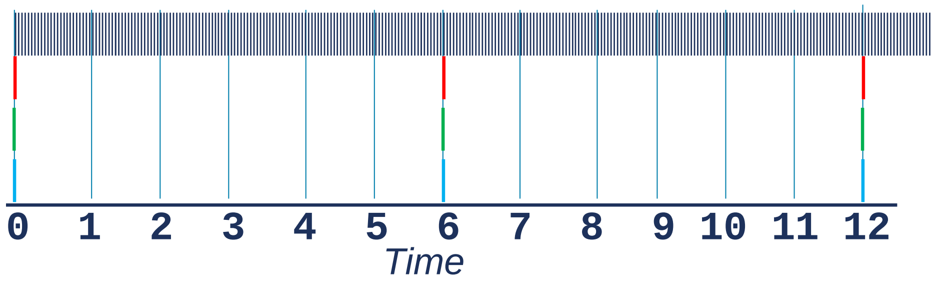
The goal of this program is to drive the LEDs through a sequence of colours slowly (red, blue, green, repeat). When the user presses a switch, the system should quickly flash all LEDs as long as the switch is pressed. The time delay between the user *pressing* the switch and the *seeing* the LED flash is the system's response time, which is very dependent on the type of scheduler used.

First make sure that you disable the round-robin policy (in RTX_Config.c) so that we can have complete control over the scheduling policy.

1.1 Software

There are four software tasks in the system.

- Three tasks control the LEDs (one LED per task). Each of these LED tasks lights its LED, executes a busy wait loop (simulating a long computation), turns off the LED, and yields the processor. In the systems with prioritisation, the red LED task has the highest priority, then the blue LED task, and then finally the green LED task.
- One task (PS) polls the blue (user) switch and flashes the LEDs as long as the switch is pressed. When the switch is released, the task yields the processor. When the LEDs are flashing this indicates that this task (PS) is running and has detected the switch has been pressed.



Task	Execution Time C_i	Period T_i
PS	Negligible, or else as long as switch is pressed	1 ms
R	1 s	6 s
G	1 s	6 s
B	1 s	6 s

Figure 1. Ideal task release schedule for PS, R, G and B.

The ideal task schedule is shown in Figure 1. The following timing behaviour is used for the prioritised cases:

- The poll switch (PS) task is scheduled to run very frequently (about every 1 millisecond) to detect user input.
- The LED tasks are scheduled to be released periodically (about every 6 seconds) and light the LED for about a second.
- Initially all tasks are released simultaneously.

The non-preemptive, non-prioritised approach runs each task in a fixed order (PS, R, B, G) as fast as possible.

You will compare how long it takes the system to start flashing the LED after you press the switch using three separate scheduling approaches.

- We will use a superloop for **non-preemptive non-prioritised** scheduling. In this case a simple infinite loop calls the four task functions sequentially.
- The RTX kernel can emulate **non-preemptive prioritised** scheduling. This is done by having a task raise its priority to the maximum level when it starts running, and then restore its original priority upon completion.
- The RTX kernel provides **preemptive prioritised** scheduling. You will evaluate task response time for two cases: when the switch task has **lower priority** than the LED tasks, and when it has **higher** priority.

For convenience, we use a wrapper function around each function to create a task which the RTOS can manage. We could eliminate the wrapper function, but then we would need to modify the internals of the task function to switch between schedulers.

Also make sure the Timer clock value in the RTX_Config.c is correct for your board. The timing behaviour may not be precise but the LED sequence should be the same for all boards, so focus on that.

2 Procedure

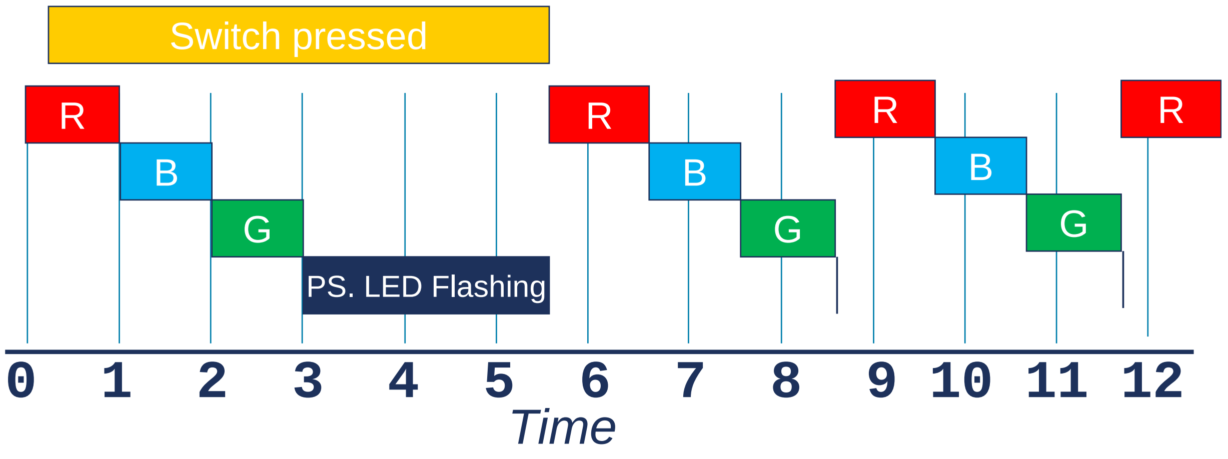
Select the scheduling approach by setting one of these symbols to 1: SCHED_NPRE_NPRI, SCHED_NPRE_PRI, SCHED_PRE_PRI.

2.1 non-preemptive non-prioritiSed

1. Open the template project. Scan through the code.
2. Configure the code to use **non-preemptive non-prioritised** scheduling (By defining SCHED_NPRE_NPRI as 1, and the others as zero). Build the project and download the executable to the MCU. Press the user switch (blue) as soon as the red LED lights up and observe the response. Complete the table below to show the LED activity before and during pressing the switch.

Switch	open	closed (pressed)		
LED Activity	red	blue	green	flashing

3. Which tasks ran between when you closed the switch and when the LED started flashing? Did they run for their full duration or did they stop early (i.e. were they pre-empted)? Explain how the scheduler affected this. What controls the order in which the tasks run?
   The tasks for red, blue and green ran (without preemption) and then the PS task could run. The scheduler uses a fixed schedule without task preemption. The tasks ran in the order R B G PS because the superloop calls them in order PS R B G and then repeats. Pressing the switch during the red task means that PS doesn't get to run until the next time through the superloop.
4. Draw a timeline showing processor activity corresponding to your LED chart.

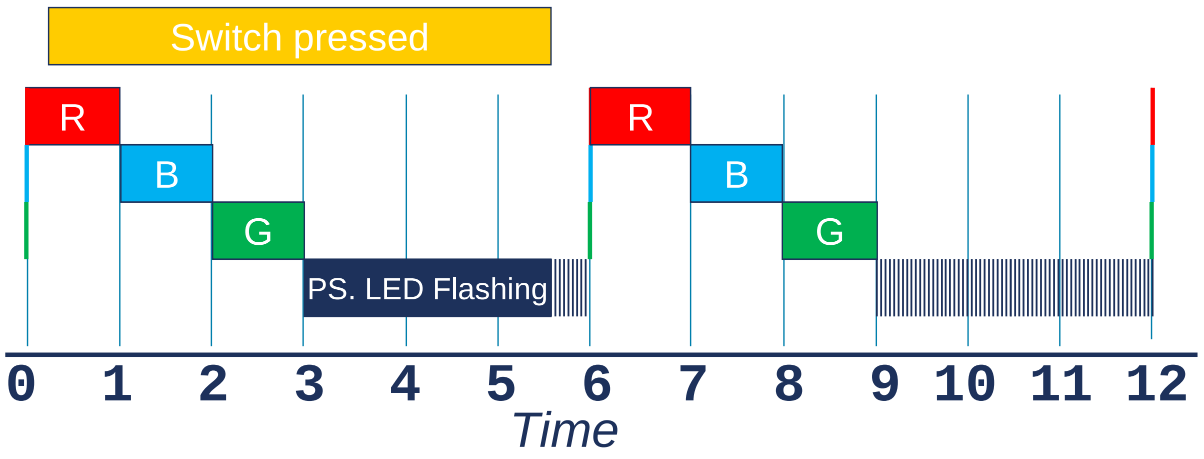


2.2 Non-preemptive prioritised

5. Configure the code to use **non-preemptive prioritised** scheduling (SCHED_NPRE_PRI as 1). Configure the task Poll_Switch to have **lower priority** than the LED tasks by defining PS_PRIORITY as 1. Build the project and download the executable to the MCU. Press the switch as soon as the red LED lights and observe the response. Complete the chart below to show the LED activity before and during pressing the switch.

Switch	open	closed (pressed)
LED Activity	green red	blue green flashing

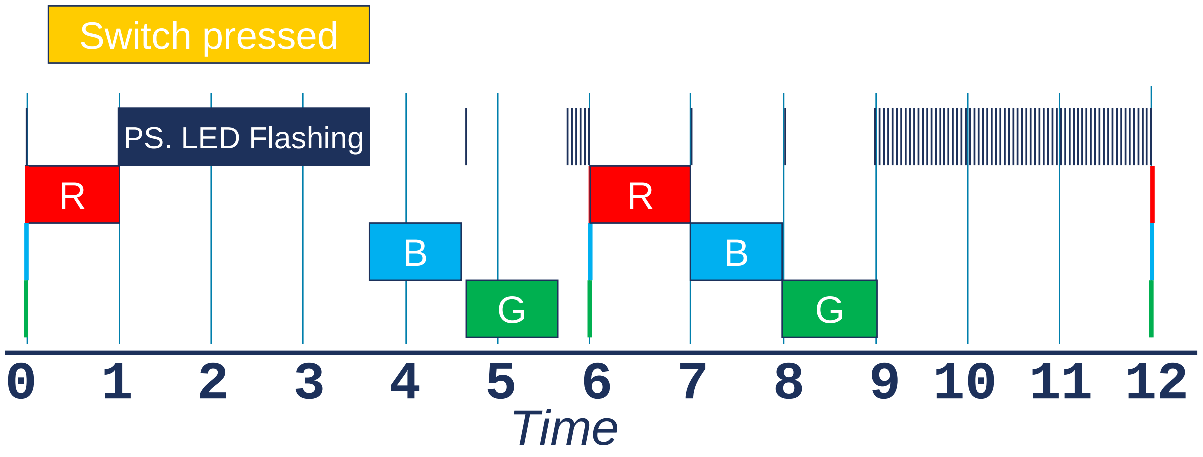
6. Which tasks ran between when you closed the switch and when the LED started flashing? Did they run for their full duration or did they stop early (i.e. were they preempted)? Explain how the scheduler affected this. What controls the order in which the tasks run?
The tasks for red, blue and green ran (without preemption) and then the PS task could run. The scheduler uses priority to select the task but doesn't use task preemption. The tasks ran in the order R B G PS because their priorities are in that order, starting with the highest (R) through the lowest (PS). Pressing the switch during the red task means that PS doesn't get to run until all tasks with priority higher than PS have run.
7. Draw a timeline showing processor activity corresponding to your LED chart.



8. Configure the code to use **non-preemptive prioritised** scheduling. Configure the task Poll_Switch to have **higher priority** than the LED tasks by defining PS_PRIORITY as 5. Build the project and download the executable to the MCU. Press the switch as soon as the red LED lights and observe the response. Complete the chart below to show the LED activity before and during pressing the switch.

Switch	open	closed (pressed)
LED Activity	green red	flashing

9. Which tasks ran between when you closed the switch and when the LED started flashing? Did they run for their full duration or did they stop early (i.e. were they preempted)? Explain how the scheduler affected this. What controls the order in which the tasks run?
   The task for red ran (without preemption) and then the PS task could run. The scheduler uses priority to select the task but doesn't use task preemption. Although task PS has the highest priority, it doesn't get to run until task R finishes and the scheduler picks the next task to execute.
10. Draw a timeline showing processor activity corresponding to your LED chart.
   N

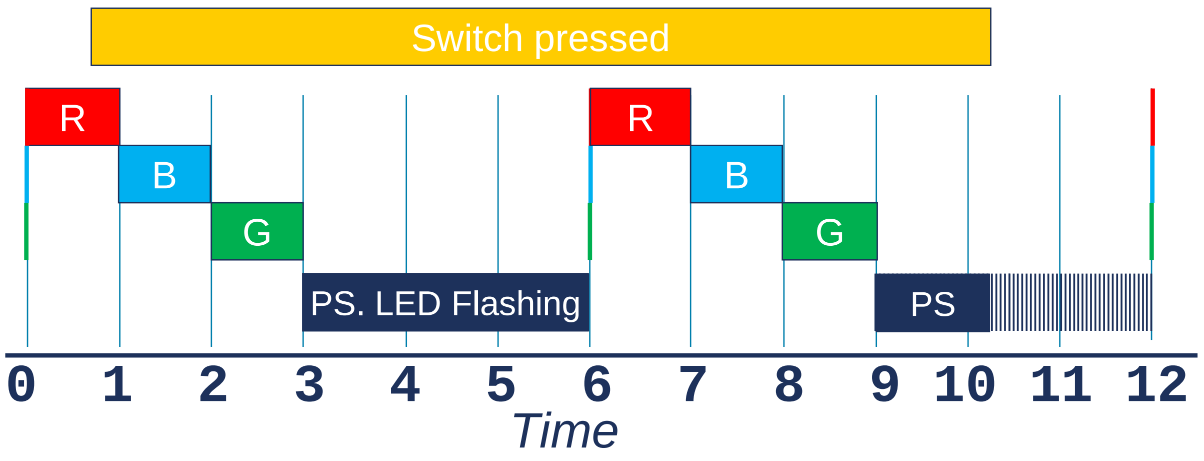


2.3 Preemptive prioritised

11. Configure the code to use **preemptive prioritised** scheduling. Configure the task Poll_Switch to have **lower priority** than the LED tasks by defining PS_PRIORITY as 1. Build the project and download the executable to the MCU. Press the switch as soon as the red LED lights and observe the response. Complete the chart below to show the LED activity before and during pressing the switch.

Switch	open	closed (pressed)
LED Activity	off red	blue green flashing

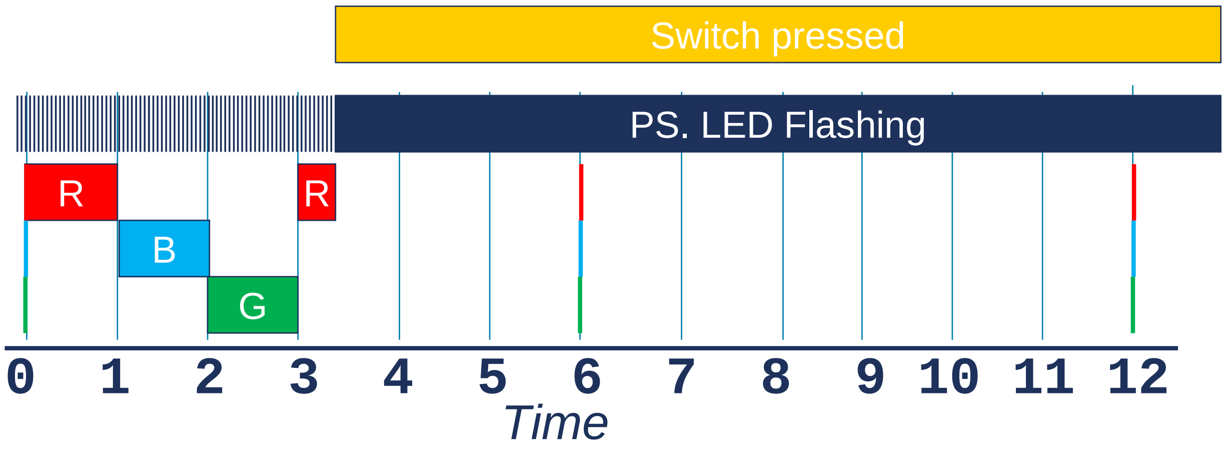
12. Which tasks ran between when you closed the switch and when the LED started flashing? Did they run for their full duration or did they stop early (i.e. were they preempted)? Explain how the scheduler affected this. What controls the order in which the tasks run?
   The tasks for red, blue and green ran (without being preempted) and then the PS task could run. The scheduler uses priority to select the task and uses task preemption. Even though we are using preemptive scheduling, the PS task has lower priority than R, G, or B, so the scheduler does not run it until R G and B have finished. The tasks ran in the order R B G PS because their priorities are in that order, starting with the highest (R) through the lowest (PS). Pressing the switch during the red task means that PS doesn't get to run until all tasks with priority higher than PS have run.
13. Draw a timeline showing processor activity corresponding to your LED chart.



14. Configure the code to use **preemptive prioritised** scheduling. Configure the task Poll_Switch to have **higher priority** than the LED tasks by defining PS_PRIORITY as 5. Build the project and download the executable to the MCU. Press the switch as soon as the red LED lights and observe the response. Complete the chart below to show the LED activity before and during pressing the switch.

Switch	open	closed (pressed)
LED Activity	off red	flashing

15. Which tasks ran between when you closed the switch and when the LED started flashing? Did they run for their full duration or did they stop early (i.e. were they preempted)? Explain how the scheduler affected this. What controls the order in which the tasks run?
All running tasks are now being preempted every 1 ms by the PS task. If the switch is not pressed, though, PS returns very quickly without changing the LED, so we do not see any effect on the system. When the switch is finally pressed we see the LED flashing immediately. In fact this is not immediate, but instead a delay of up to 1 ms (the period at which PS runs). In order for a preemptive system to work, we need to allocate the priorities correctly.
16. Draw a timeline showing processor activity corresponding to your LED chart.


Switch pressed
PS. LED Flashing
R
B
G
R
0 1 2 3 4 5 6 7 8 9 10 11 12
Time