Prioritized Pre-emptive Preemptive multitasking

- Preemptive multitasking is a form of multitasking where the operating system interrupts and switches between tasks, or threads, without the tasks' cooperation.
- Along with Prioritized preemption, time slicing is also implemented.
- In preemptive multitasking systems, the operating system has the ability to forcefully interrupt a currently running task and switch to another task, typically based on priority and time-sharing considerations.
- Refer to Preemptive Scheduling in Mastering the FreeRTOS™ Real Time Kernel page 92.

Cooperative multitasking

- FreeRTOS primarily employs preemptive scheduling, it also supports cooperative multitasking to some extent.
- Cooperative multitasking means that tasks voluntarily yield control to allow other tasks to run, rather than being preempted by the scheduler.
- Refer to Co-operative Scheduling in Mastering the FreeRTOS™ Real Time Kernel page 98.

Prioritized Pre-emptive Scheduling with Time Slicing and Co-operative Scheduling

- Refer to above topics in Mastering the FreeRTOS™ Real Time Kernel page 92 onwards.
- 10 Mins of Reading ☺
- Can ask random students to come and explain the same ☺

Assignment Problem 3

Scenario 1 (Pre-emptive Scheduling)

Create and RTOS application with following behaviour.

- Create Task1 with priority 2 for toggling LED1.
- Create Task2 with priority 2 for toggling LED2.
- Start scheduler, observe the behaviour and document.

Scenario 2 (Co-operative Scheduling)

Create and RTOS application with following behaviour.

- Set **#define configUSE_PREEMPTION 0 in FreeRTOSConfig.h**
- Create Task1 with priority 2 for toggling LED1.
- Create Task2 with priority 2 for toggling LED2.
- Start scheduler, observe the behaviour and document.

Cooperative multitasking

- Task Yielding()

  Tasks can explicitly yield the processor using the taskYIELD function. When a task calls taskYIELD, it voluntarily gives up its CPU time, allowing other tasks of equal or higher priority to run.

- Cooperative Blocking

  Tasks can be designed to cooperatively block themselves using functions like vTaskSuspend or vTaskDelay. When a task is blocked, it allows other tasks to run until it becomes unblocked.

- Task Notification

  Task notifications can also be used for cooperative scheduling. A task can wait for a notification and voluntarily unblock when the notification is received.

Assignment Problem 4

Scenario 1

Create and RTOS application with following behaviour.

- Create Task1 with priority 2 for toggling LED1 at every 100 ms.
- Create Task2 with priority 2 for toggling LED2 at every 500 ms.
- Start scheduler, observe the behaviour and document.

Scenario 2

Create and RTOS application with following behaviour.

- Set **#define configUSE_PREEMPTION 0 in FreeRTOSConfig.h**
- Create Task1 with priority 2 for toggling LED1 at every 100 ms.
- Create Task2 with priority 2 for toggling LED2 at every 500 ms.
- Start scheduler, observe the behaviour and document.

Assignment Problem 5

Scenario 1

Create and RTOS application with following behaviour.

- Set **#define configUSE_PREEMPTION 0 in FreeRTOSConfig.h**
- Create Task1 with priority 2 for toggling LED1 at every 100 ms and call **taskYIELD();**
- Create Task2 with priority 2 for toggling LED2 at every 500 ms and call **taskYIELD();**
- Start scheduler, observe the behaviour and document.

Scenario 2

Create and RTOS application with following behaviour.

- Set **#define configUSE_PREEMPTION 0 in FreeRTOSConfig.h**
- Create Task1 with priority 1 for toggling LED1 at every 100 ms and call **taskYIELD();**
- Create Task2 with priority 2 for toggling LED2 at every 500 ms and call **taskYIELD();**
- Start scheduler, observe the behaviour and document.