Comp3402 Real-Time Operating System (RTOS)

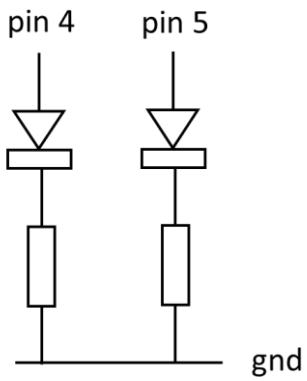
C.B.Price March 2023

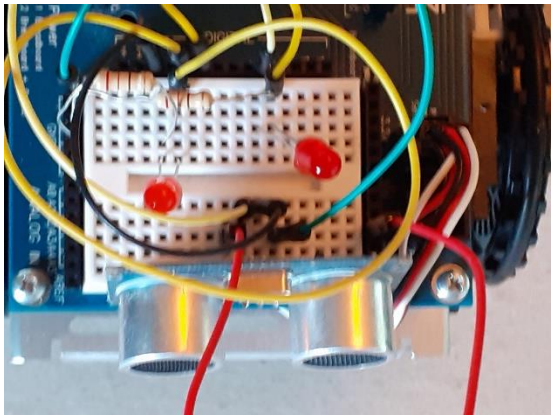
Purpose	(i) To investigate basic concepts of the RTOS (ii) To investigate the Scheduler
Files Required	Arduino **portable** sketchbook
ILO Contribution	6
Send to Me	
Assignment Info.	
Homework	Read Chapter 8 (10)

Activities

1 Two Tasks – Blinking LEDs

(a) Wire up the following circuit on the Robot protoboard. The resistors are 220 Ohm (Red Red Brown) and the short leg of the LED should be connected to the resistor.





(b) Open the sketch **RTOS_Rel_1** (File -> Sketchbook). You will see one task has been created like this

xTaskCreate(MyTask1, "Task1", 200, NULL, 1, NULL);

The arguments are, from left to right (i) the name of the task function which appears in the code below, (ii) a human-readable name, (iii) size of the stack memory for the task, (iv) a NULL pointer, (v) the task priority, (vi) another NULL pointer.

You will also see the code for **MyTask1** which comprises an un-ending while loop which contains code to switch the LED on pin 4 HIGH and LOW with a total period of 2000 milliseconds (2 seconds). Upload and run the sketch and see what happens.

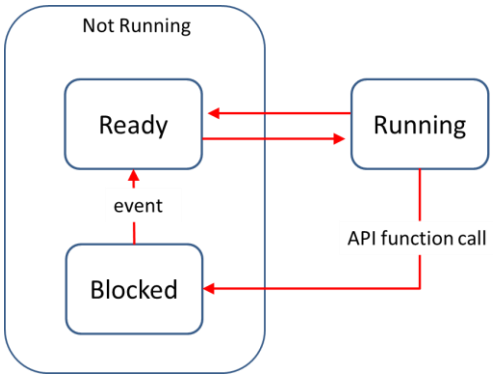
(c) Add a line of code to create a second task with the same priority (and stack size) Now add the body of the second task to switch the LED on pin 5 HIGH and LOW. You can choose the time.

(d) Compile and upload. You should see both LEDs flashing, one being driven by each task. Remember that the tasks are being swapped in and out around 60 times per second.

(e) Now raise the priority of one of the tasks from 1 to 2. Compile and upload. Observe, what's happening here? Explain why!

2 Making use of the Blocked state

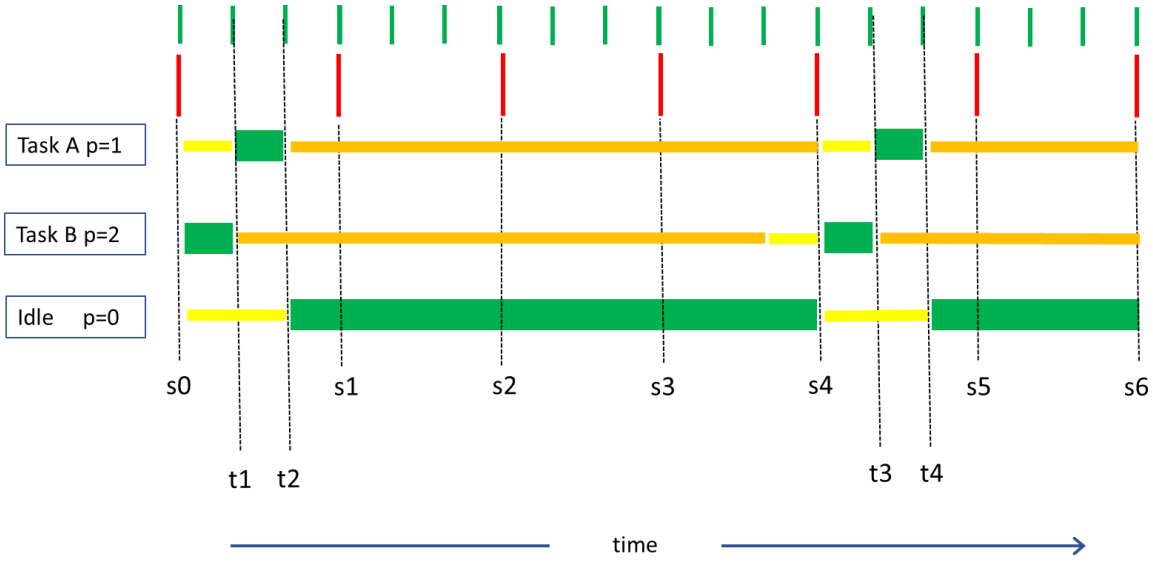
In the above example, the tasks were hogging the CPU, starving it of cycles to correctly schedule the tasks. The scheduler needs to be placed into the Blocked state so its Idle thread can run which gives it opportunity to swap in a lower priority task. Here we shall investigate this.



(a) Run the Sketch **RTOS_Rel_2** and open the serial monitor. The task grabs the number of times the Scheduler has entered the Blocked state and prints this out. You should see there are no idle calls, so the scheduler is being starved.

(b) Replace all the **delay(1000);** function calls with **vTaskDelay(1000/portTICK_PERIOD_MS);** This function places the calling task into the blocked state for a number of ticks which will allow the Scheduler to swap any other Ready task in. You should now see a huge number of idle task calls.

(c) Increase the priority of one of the tasks. Now you should see that the Scheduler can now swap both of them in and out. Compare this with the behaviour of 1(f). Here's a timing diagram for this situation. Make sure you understand this.



(d) In this example, one task had a higher priority. From the above diagram can you work out which this was?