Neat Example

	priority	description
Task 3	2	periodic task, prints string
Task 1	1	continuous task, prints string
Task 2	1	continuous task, prints string

```
/* continuous task */
void vContinuousProcessingTask( void *pvParameters)
{
char *pcTaskName;
    pcTaskName = (char *) pvParameters;
    for ( ;; )
    {
      vPrintString (pcTaskName);
    }
}
/* periodic task */
void vPeriodicTask(void *pvParameters)
{
TickType_t xLastWakeTime;
const TickType_t xDelay3ms = pdMS_TO_TICKS( 3 );
    xLastWakeTime = xTaskGetCount( );
        /* retrieves current tick time and stores it,
          which corresponds to the time this task
          enters running state --- this variable is
          subsequently updated by vTaskDelayUntil() */
    for ( ;; )
    {
        vPrintString ("Periodic task is running\r\n");
        vTaskDelayUntil ( &xLastWakeTime, xDelay3ms );
          /* task executes every 3ms exactly */
    }
}
```

- main code below:

```
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\r\n";
static const char *pcTextForTask3 = "Periodic task is running\r\n";
int main (void)
{
    xTaskCreate (vContinuousProcessingTask, "Task 1", 1000,
                 (*void) pcTextForTask1, 1, NULL);

    xTaskCreate (vContinuousProcessingTask, "Task 2", 1000,
                 (*void) pcTextForTask2, 1, NULL);

    xTaskCreate (vPeriodicTask, "Task 3", 1000
                 (*void) pcTextForTask3, 2, NULL);

    vTaskStartScheduler();
    return 1;
}
```

(stack size → 1000)

→ should produce an output like:

```
Task 2 is running
Task 2 is running
Periodic task is running
Task 1 is running
Task 1 is running
Task 1 is running
Task 2 is running
Task 2 is running
Task 2 is running
Task 2 is running
Task 1 is running
Task 1 is running
Task 1 is running
Task 1 is running
Periodic Task is running
   :
   :
```

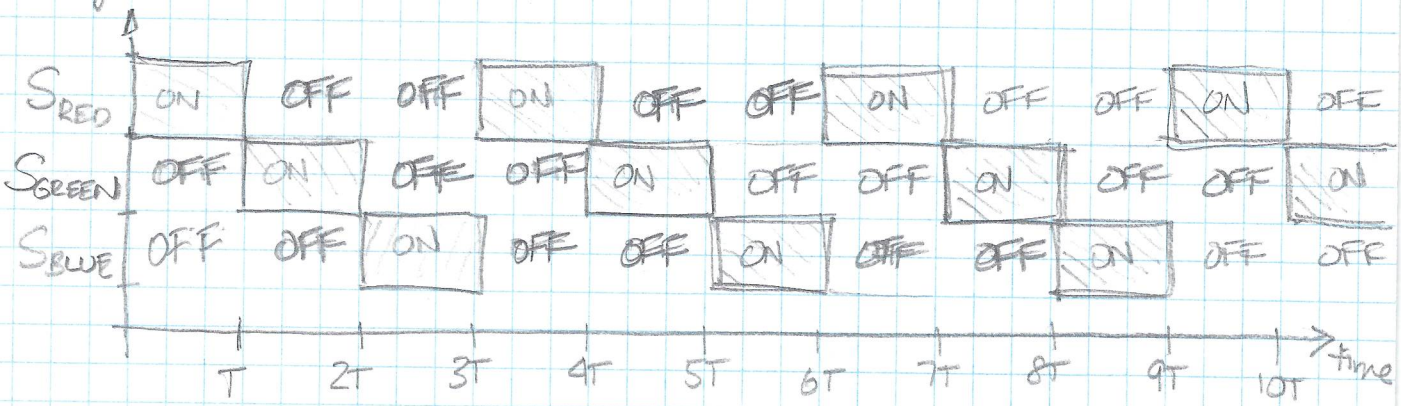
} Task 1 prints the string a number of times while it is in the running state

} ... as does Task 2 ...

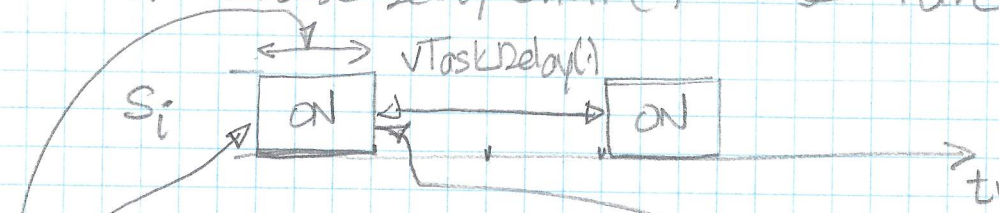
both Tasks remain in the running state for a full tick period, during which the text may be printed a number of times...

Breaking Down the Revised FreeRTOS Blinky Assignment

- Recall that in this assignment you are to use the FreeRTOS framework to create a real-time implementation of sequential blinking, in which the Red, Green and Blue LEDs flash with no overlapping.
- Just like the conceptual work we have done with RTOS algorithms like RM and EDF we can construct a timing diagram for this assignment:



- in order to realize the "OFF" time for any of the LEDs, we make the task go into its BLOCKED state, using either the vTaskDelay() or vTaskDelayUntil() API functions.



during this time, regardless of the priority of the task, other tasks may execute.

- also note that each LED task is executing the same sequence of events, namely ON, OFF, OFF, but out of phase w.r.t. each other. For a given task, this sequence might be written as

```
void LEDTask( void* pvParameters ) {
  ⋮
  while(1) {
    LEDon(LED_id);
    delay_function(T);
    LEDOff(LED_id);
    vTaskDelay(T);
    vTaskDelay(T);
  }
}
```

Can you suggest how to arrange this loop for the Green and Blue LEDs?

- note that delay_function(T) can be realized using either vTaskDelay() / vTaskDelayUntil() or with a private function that does not put the calling task in the Blocked state. Why?

- there are actually many ways to realize this application!

- we can also envision each task as being preempted by the others, without going into a blocked state, but because of changing priorities
- Each task can be implemented by the same task function:

```
void TaskFunction(void * pvParameters)
{
    :
    LEDOff(LED_id);
    while (1)
    {
        LEDOn(LED_id);
        delay_function(T);
        LEDOff(LED_id);
        delay_function(2*T);
    }
}
```

- along with a timer task that keeps track of when to rotate task priorities