Introduction to FreeRTOS

ENCE464 Embedded Software and Advanced Computing

Course coordinator: Steve Weddell (steve.weddell@canterbury.ac.nz)
Lecturer: Le Yang (le.yang@canterbury.ac.nz)
Department of Electrical and Computer Engineering

Where we’re going today

- **Task scheduler vs RTOS**
- Example code using FreeRTOS
- FreeRTOS terminology

Task Scheduler vs RTOS (1)

- Task scheduler
  - Tasks always return to scheduler
  - Programmers are responsible for generation and management of tasks
  - Simple task scheduler takes little less resources than RTOS
  - Examples: non-preemptive, preemptive, time-slice …

Task Scheduler vs RTOS (2)

- Real-time operating system (RTOS)
  - RTOS performs task switch
  - Typically used for large embedded systems to reduce complexity
    - Provides programmers with a higher level of abstraction
  - More controls for event and task synchronization
  - Programmers just write and maintain tasks → less maintenance

Where we’re going today

- Task scheduler vs RTOS
- **Example code using FreeRTOS**
- FreeRTOS terminology

FreeRTOS: a First Look (1)

- From www.freertos.org
  - “FreeRTOS is a market leading real time operating system (or RTOS) from Real Time Engineers Ltd. that supports 34 architectures and receives **107000** downloads a year”
  - “It is professionally developed, strictly quality controlled, robust, supported, and free to use in commercial products without any requirement to expose your proprietary source code. Why would you choose anything else?”

FreeRTOS: a First Look (2)

- From www. Freertos.org
  - FreeRTOS is designed to be simple and easy to use: Only 3 source files that are common to all RTOS ports, and one microcontroller specific source file are required, and its API is designed to be simple and intuitive.

  - It supports
    - Tasks and co-routines
    - Tasks are implemented as C functions
    - Fixed priority pre-emptive scheduling
    - Optional cooperative scheduling
    - Queues
    - Binary and counting semaphores
    - Mutexes and recursive mutexes
    - Stack overflow detection

Example Code (1)

```
int main(void)
    {
        // Initialise the required peripherals.
        InitStatusLight();                /*  .   .   .   .   */

        /* Create the queue used by the OLED task.  Messages for display
             on the OLED are received via this queue. */
        xSendQueue  = xQueueCreate(mainSEND_QUEUE_SIZE,sizeof(xQueueMessage));

        /*-------------------------------------------
             Create tasks and start scheduler
        -------------------------------------------*/

        /* Create the required tasks */
        xTaskCreate(vSendTask, "Send Task", 240, NULL, 1, NULL);
        xTaskCreate(vLedBlink, "LED Blink", configMINIMAL_STACK_SIZE,
                     NULL, 4, NULL);
        vStartControlTasks(xSendQueue);

        // Enable interrupts to the processor.
        IntMasterEnable();

        /* Start the scheduler so our tasks start executing. */
        vTaskStartScheduler();

        /* If all is well we will never reach here (scheduler running).   */
        while (1)
        {
        }
    }
```

Example Code (2)

```
/**************************************************************************
 * Pop messages from the queue and send to the OLED display.              *
 **************************************************************************/
static void
vSendTask( void *pvParameters )
{
   /*   .    .    .    .   */
   for( ;; )
   {
      /* Wait for a message to arrive that requires displaying. */
      xStatus = xQueueReceive( xSendQueue, &xMessage, portMAX_DELAY );

      if (xStatus == pdPASS)
      {
         switch (xMessage.type)
         {
            case (CURRENT_ALTITUDE):
                     sprintf(cMessage, "Current Alt: %d", xMessage.pcMessage);
                          break;

            /*  Other cases  */

                  case (PWM_DUTY):
                     sprintf(cMessage, "PWM Duty: %d", xMessage.pcMessage);
                        break;
                  default:
                     sprintf(cMessage, "Other: %d", xMessage.pcMessage);
         }
         RIT128x96x4StringDraw(cMessage, 0, ulY, 8);
      }
   }
}
```

Where we’re going today

- Task scheduler vs RTOS
- Example code using FreeRTOS
- **FreeRTOS terminology**

Running and Not Running Tasks

- **Tasks** are used in preference of **thread** to represent the sequence of instructions independently managed by FreeRTOS
  - Each task has its own stack initialized by FreeRTOS when created
- Tasks are either in state Running or Not Running (the latter contains 3 sub-states)

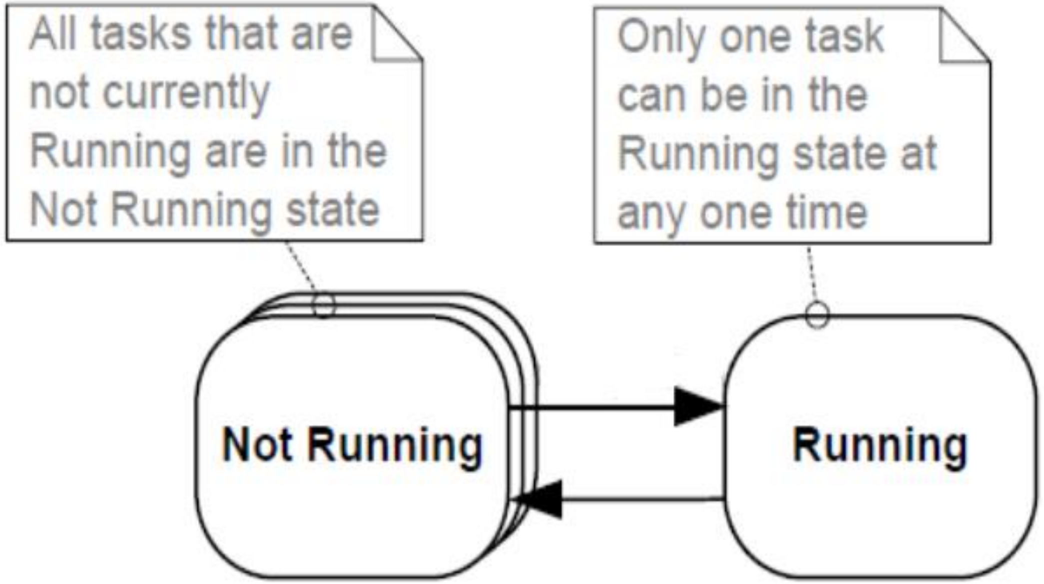
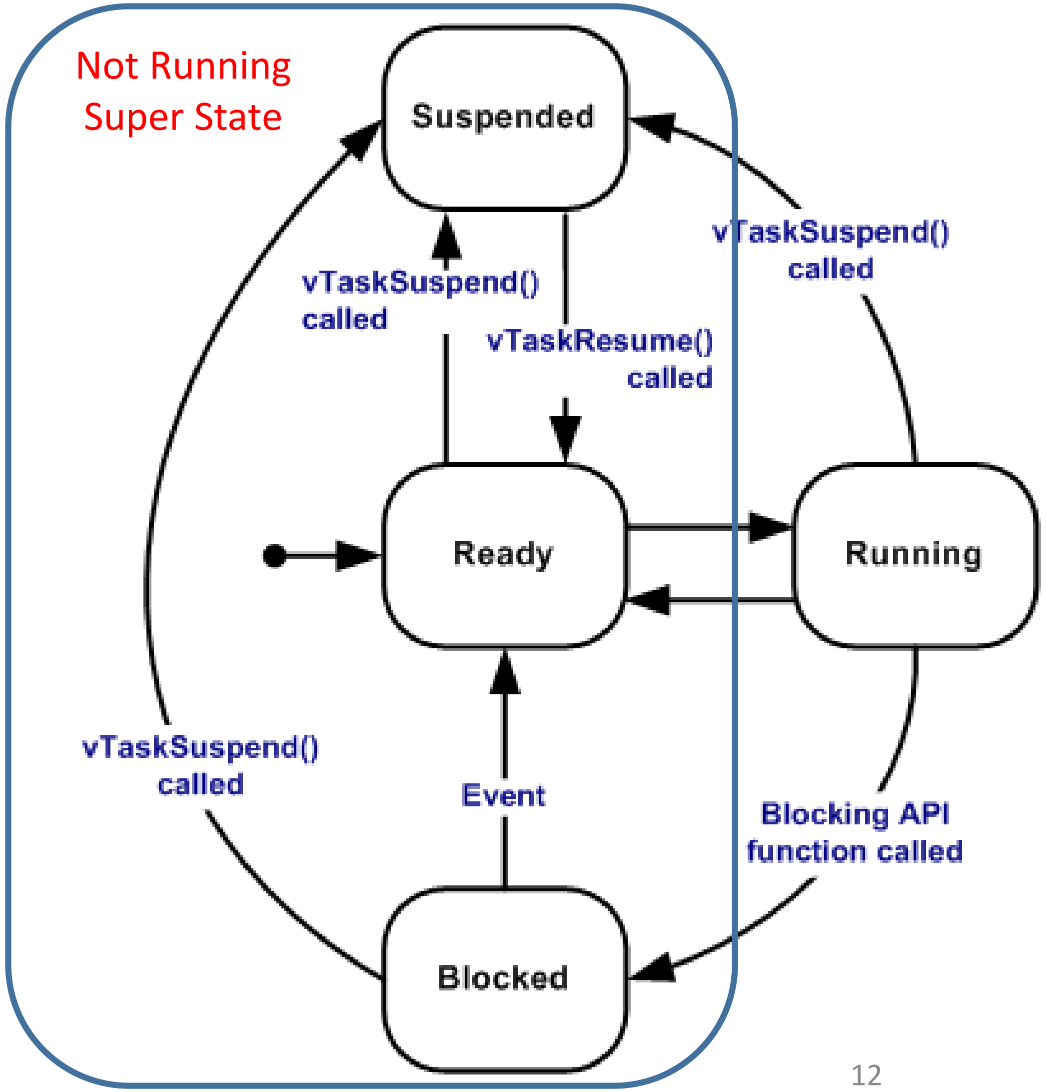


Figure 1. Top level task states and transitions

Ready, Suspended and Blocked Tasks

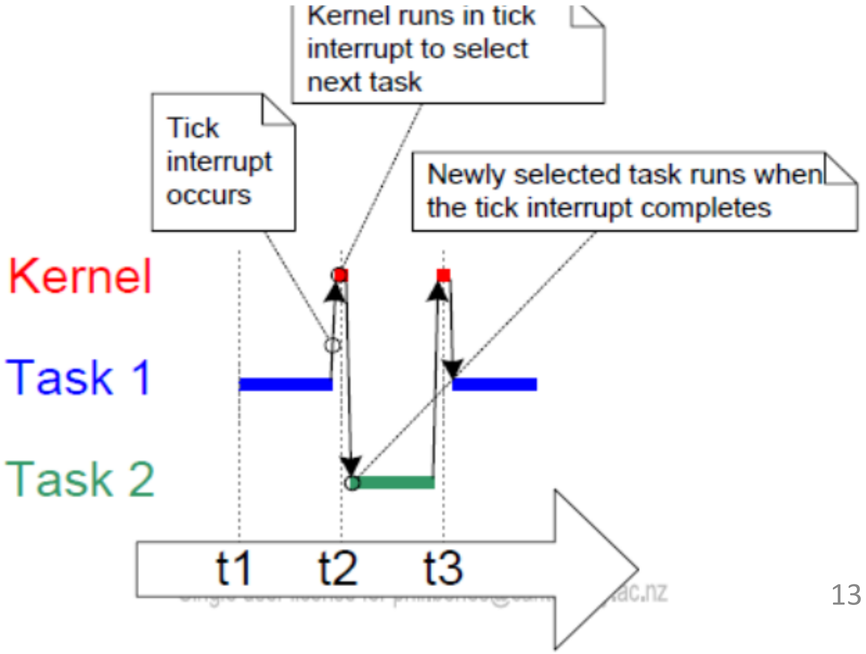
- Ready
  - Tasks that are able to run but are not running
- Blocked
  - Tasks that are currently waiting for either a temporal (e.g., synchronization) or external event (e.g., I/O operation)
- Suspended
  - Tasks that are stopped and resumed by OS due to e.g., preemption or time-slicing



FreeRTOS Scheduler

- FreeRTOS runs a round-robin pre-emptive scheduler (kernel)
  - Tasks with lower priority numbers have low priorites (the opposite to Cortex M4)

- Perform task switches on ticks (controlled by SysTick interrupts)
  - configTICK_RATE_HZ sets time slicing rate
    - 100 Hz for a 10 ms time slice for the scheduler



Hook Functions

- Hook functions are callback functions
  - E.g., Idle task can optionally call a user-defined hook function – the idle hook

```
void vApplicationIdleHook( void )
{
volatile size_t xFreeStackSpace;

    /* The idle task hook is enabled by setting configUSE_IDLE_HOOK to 1 in
    FreeRTOSConfig.h.

    This function is called on each cycle of the idle task.  In this case it
    does nothing useful, other than report the amount of FreeRTOS heap that
    remains unallocated. */
    xFreeStackSpace = xPortGetFreeHeapSize();

    if( xFreeStackSpace > 100 )
    {
        /* By now, the kernel has allocated everything it is going to, so
        if there is a lot of heap remaining unallocated then
        the value of configTOTAL_HEAP_SIZE in FreeRTOSConfig.h can be
        reduced accordingly. */
    }
}
```

  - Other possible uses for idle hook: measuring spare processor capacity
  - tick hook, stack overflow hook and malloc failed hook functions