Course Evaluation

- Please take this time to fill out the course evaluation
- We're working on updating the labs and other aspects of the course and really need your feedback to improve
- We haven't gotten much feedback the past few quarters and it's been really hard to improve

https://uw.iasystem.org/survey/290587

Lecture 17: FreeRTOS examples

Vikram Iyer

Adapted from material by Blake Hannaford

Announcements

Memorial Day 27	13:00-15:00 OH (Zach) 28 ECE 345	9:30-11:30 OH (Deeksha) 29 ECE 345 12:30-14:20 Lecture MOR 230 *Lecture 17: FreeRTOS Examples* 12:30-15:30 Quiz (20 min during lecture) MOR 230 (Lecture room) 14:00-15:00 OH (Vikram) ECE 345 14:00-15:00 Quiz (DRS, during OH) ECE 345 (Lab room)	10:00-12:00 OH (Zach) 30 ECE 345 12:30-14:30 OH (Alex) ECE 345	12:30-15:00 OH (Vikram) 31 ECE 345

June

Monday	Tuesday	Wednesday	Thursday	Friday
9:30-11:30 OH (Deeksha) 03 ECE 345 11:30-13:30 OH (Alex) ECE 345	13:00-15:00 OH (Zach) 04 ECE 345	9:30-11:30 OH (Deeksha) 05 ECE 345 14:00-15:00 OH (Vikram) ECE 345	10:00-12:00 OH (Zach) 06 ECE 345 12:30-14:30 OH (Alex) ECE 345 23:59 Lab 4 due	14:00-15:00 OH (Vikram) 07 ECE 345

Creating a task

```
void vTaskCode( void * pvParameters ) { // Actual task definition
    /* The parameter value is expected to be 1 as 1 is passed in the pvParameters value in the
       call to xTaskCreate() below. configASSERT( ( ( uint32_t ) pvParameters ) == 1 );
    for( ;; ) {
        /* Task code goes here like in a loop() function. */
    }
}
```

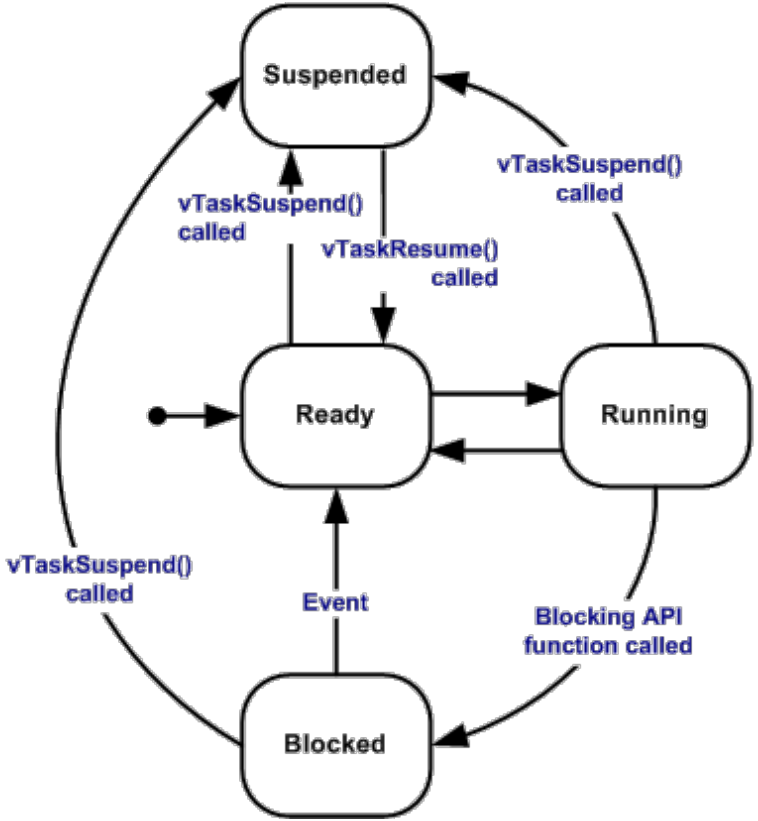
```
void setup() {
    BaseType_t xReturned;
    TaskHandle_t xHandle = NULL;
    xReturned = xTaskCreate(          /* Create the task, storing the handle. */
                    vTaskCode,        /* Function that implements the task. */
                    "NAME",           /* Text name for the task. */
                    STACK_SIZE,       /* Stack size in words, not bytes. */
                    ( void * ) 1,     /* Parameter passed into the task. */
                    tskIDLE_PRIORITY,/* Priority at which the task is created. */
                    &xHandle );       /* Used to pass out the created task's handle. */

    if( xReturned == pdPASS ){
        /* The task was created.  Use the task's handle to delete the task. */
        vTaskDelete( xHandle );
    }
}
```

Creating a task

```
void TaskBlink(void *pvParameters)
    for (;;) {
        digitalWrite(LED_PIN, HIGH); // turn the LED on (HIGH is the voltage level)
        vTaskDelay( 500 / portTICK_PERIOD_MS ); // wait for one second
        digitalWrite(LED_PIN, LOW); // turn the LED off by making the voltage LOW
        vTaskDelay( 500 / portTICK_PERIOD_MS ); // wait for one second
    }
}
```

```
void TaskBlink( void *pvParameters );

void setup() {
    xTaskCreate(
        TaskBlink,
        "Blink”, // A name just for humans
        128,     // This stack size in words (16b)
        NULL,    // Parameters
        2,       // Priority, with 3
                 //(configMAX_PRIORITIES - 1)
                 // being the highest, and 0
                 // being the lowest.
    , NULL );
}
```

FreeRTOS task states



Running- When a task is actually executing it is said to be in the Running state. It is currently utilising the processor. There can only be one task in the Running state at any given time.

Ready- Ready tasks are those that are able to execute not currently executing because another task is using CPU

Blocked- A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event (e.g. vTaskDelay(), waiting for queue. Tasks have a 'timeout' period, after which the task will be unblocked

Suspended- These tasks do not run. Tasks only enter or exit the Suspended state when explicitly commanded to do so through the vTaskSuspend() and xTaskResume() API calls respectively.

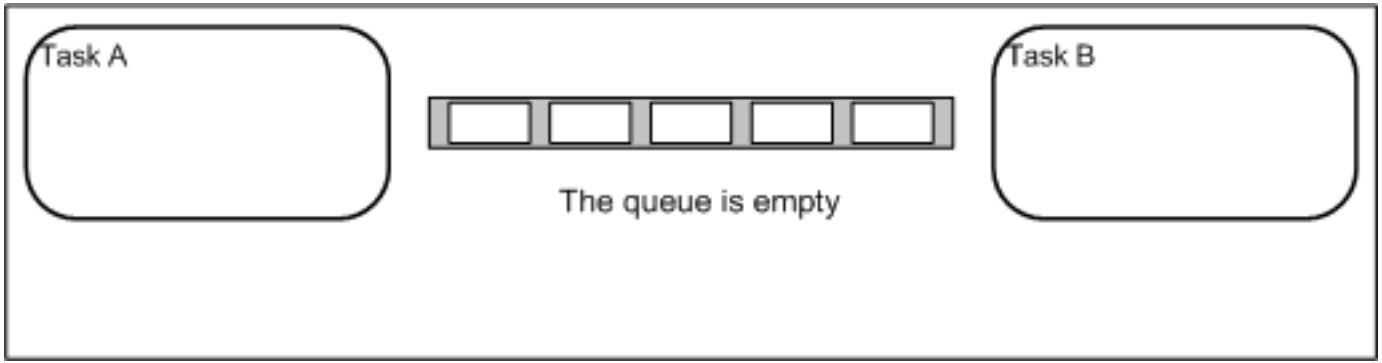
FreeRTOS timing and task control

```
vTaskDelay( 500 / portTICK_PERIOD_MS ); // wait for one second

TickType_t tick = xTaskGetTickCount();

vTaskSuspend( TaskHandle_t xTaskToSuspend );

vTaskResume( xHandle );
```

```
…
// Suspend ourselves.
vTaskSuspend( NULL );

/* We cannot get here unless
another task calls vTaskResume
with our handle as the
parameter.*/
…
```

```
// Resume the suspended task
vTaskResume( xHandle );
```

Sending data between tasks using Queues




```
// Set up the Queue
QueueHandle_t queue;

void setup(){
  queue = xQueueCreate(
      2,
      sizeof(int)
  );
}
```

```
// Task A adds to the Queue
void taskA(){
  xQueueSendToBack(
    queue,
    &data_to_send,
    1);
}
```

```
// Task A adds to the Queue
void taskA(){
  xQueueReceive(
    queue,
    &received_data,
    0);
}
```

Debugging tips

- When you have a problem first try to isolate is it a hardware problem or software problem? Then within that get more specific.

- When working with new hardware peripherals test these in isolation first. Once you get that part working and understand what that code is doing integrate it into the rest of your project

- Figure out where your code is breaking either with a print statement or for time critical things (interrupts etc) toggling and LED

- For timing issues, checking voltage etc try using the oscilloscope

- Check out FreeRTOS's debugging features
  - Checking for stack overflows/memory issues https://www.freertos.org/Stacks-and-stack-overflow-checking.html
  - Task utilities (list of tasks etc.) https://www.freertos.org/a00021.html
  - LEDs Slow blink = stack overflow. Fast blink = heap malloc() failure.