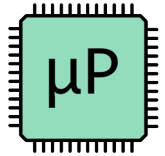
Real-time Operating Systems Examples with FreeRTOS

Lecture 24

Microprocessor-based Systems (E155)

Prof. Josh Brake



Learning Goals

- To understand the following key concepts of Real-time Operating Systems through examples in FreeRTOS
  - Task creation
  - Basic scheduling
  - Task priorities and preemption

Outline

- FreeRTOS Refresher
- Examples for today
  1. Task creation with LED blink example: `01_task_creation_blink_led.c`
  2. Passing parameters into task using pvParameters: `02_passing_parameters_blink_led.c`
  3. Multiple tasks with two serial prints: `03_multiple_tasks_print.c`
  4. Simple preemption example: poll button and blink LED and single print: `04_simple_preemption.c`

FreeRTOSConfig.h

- Configuration file used to set some of the common options for the kernel

```
// <o>Minimal stack size [words] <0-65535>
// <i> Stack for idle task and default task stack in
words.
// <i> Default: 128
#define configMINIMAL_STACK_SIZE ((uint16_t)(128))

// <o>Total heap size [bytes] <0-0xFFFFFFFF>
// <i> Heap memory size in bytes.
// <i> Default: 8192
#define configTOTAL_HEAP_SIZE ((size_t)8192)

// <o>Kernel tick frequency [Hz] <0-0xFFFFFFFF>
// <i> Kernel tick rate in Hz.
// <i> Default: 1000
#define configTICK_RATE_HZ ((TickType_t)1000)
```

```
/* Defines needed by FreeRTOS to implement CMSIS RTOS2
API. Do not change! */
#define configCPU_CLOCK_HZ (SystemCoreClock)
#define configSUPPORT_STATIC_ALLOCATION 1
#define configSUPPORT_DYNAMIC_ALLOCATION 1
#define configUSE_PREEMPTION 1
#define configUSE_TIMERS 1
#define configUSE_MUTEXES 1
#define configUSE_RECURSIVE_MUTEXES 1
#define configUSE_COUNTING_SEMAPHORES 1
#define configUSE_TASK_NOTIFICATIONS 1
#define configUSE_TRACE_FACILITY 1
#define configUSE_16_BIT_TICKS 0
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 0
#define configMAX_PRIORITIES 56
#define configKERNEL_INTERRUPT_PRIORITY 255
```

Revisiting Not Running State

- Three Options
  - Suspended
  - Ready
  - Blocked

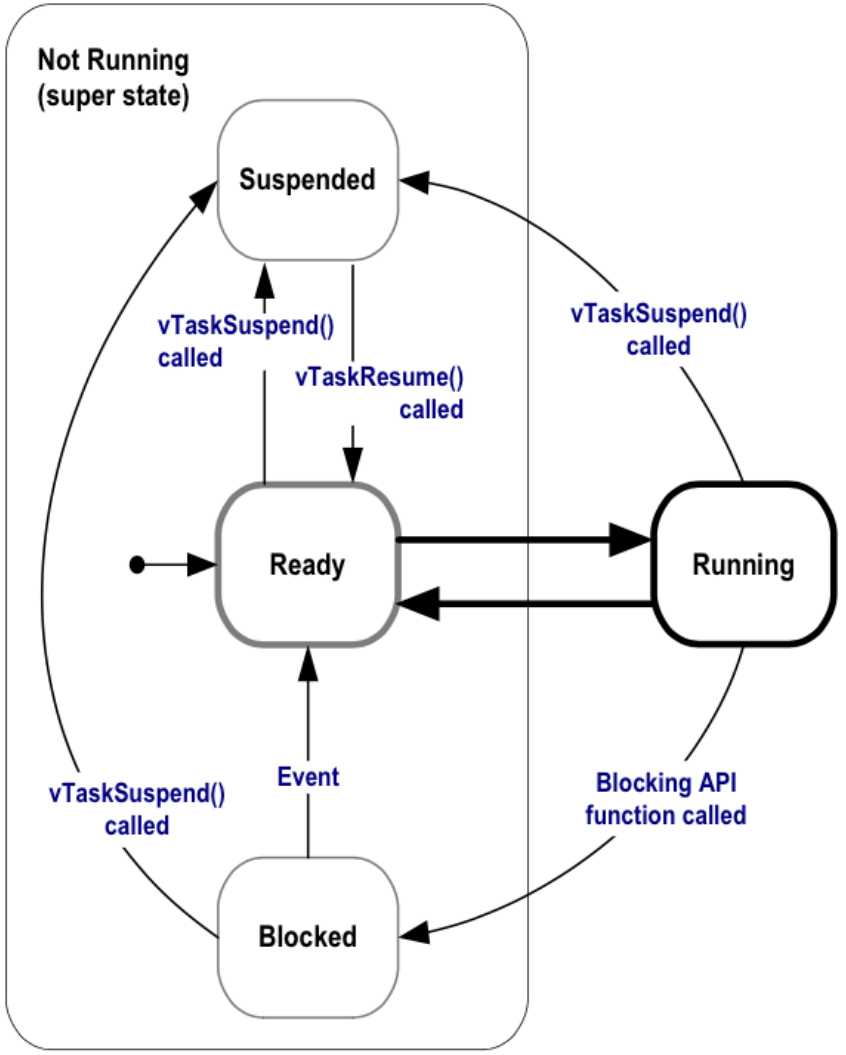


Figure 15. Full task state machine

Barry, Richard. *Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide*. 2016.

Ex. 1: Task creation with LED blink

- Basic task creation workflow
  - Set up and initialization as usual
  - Write tasks according to specified prototype
  - In main
    - Call initialization functions
    - Create tasks
    - Start scheduler

Ex. 1: Task creation with LED blink

```
// Task to toggle LED
static void toggleLedTask(void *pvParameters)
{
    const TickType_t xDelay = pdMS_TO_TICKS(500);

    while (1)
    {
        /* Simply toggle the LED every xDelay ms,
blocking between each toggle. */
        toggleLED(LED_PIN);
        vTaskDelay(xDelay);
    }
}
```

Ex. 1: Task creation with LED blink

```
// Main function where initialization is performed and tasks are created.
int main()
{
    // Call initialization functions
    init_flash();   // Set up flash
    init_clock();   // Configure 84 MHz clock rate
    init_gpio();    // Initialize GPIO for LED

    // Create tasks
    const size_t xRegTestStackSize = 250U; // Set value for stack for each task.
    xTaskCreate(toggleLedTask,      // Task function
                "Blink_1",          // Optional name for task
                xRegTestStackSize,  // Task stack size
                (void*)&led_1,      // void pointer to optional parameters
                1,                  // Task priority
                NULL);              // Handle to created task

    // Start the scheduler
    vTaskStartScheduler();

    // Infinite while loop. Should never get here unless the scheduler fails to start.
    while (1);
}
```

Ex. 2: Passing parameters into task using pvParameters

02_passing_parameters_blink_led.c

```
// New type to hold information about LED
typedef struct param_led {
   uint32_t delay_ms;
   uint8_t led_pin;
} param_led;


// Create param_led struct to hold delay and pin number for LED.
param_led led_1 = {200, 5}; // delay_ms = 200, led_pin = 5


// Task to toggle LED
static void toggleLedTask(void *pvParameters)
{
    const param_led * led_info = (param_led *) pvParameters;
    const TickType_t xDelay = pdMS_TO_TICKS(led_info->delay_ms);

    while (1)
    {
        /* Simply toggle the LED every xDelay ms, blocking between each toggle. */
        toggleLED(led_info->led_pin);
        vTaskDelay(xDelay);
    }
}
```

Ex. 3: Multiple tasks with two serial prints

```
// Initialize and configure USART
void init_uart() {
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
  RCC->APB1ENR |= RCC_APB1ENR_USART2EN;

  // Configure PA2 and PA3 as alternate functions USART2
  GPIOA->MODER  &= ~(GPIO_MODER_MODE2 | GPIO_MODER_MODE3);
  GPIOA->MODER  |= (0b10 << GPIO_MODER_MODER2_Pos | 0b10 << GPIO_MODER_MODER3_Pos);
  GPIOA->AFR[0] |= (0b0111 << GPIO_AFRL_AFSEL2_Pos | 0b0111 << GPIO_AFRL_AFSEL3_Pos);

  USART->CR1 |= (USART_CR1_UE);
  USART->CR1 &= ~(USART_CR1_M | USART_CR1_OVER8);
  USART->CR2 &= ~(USART_CR2_STOP);

  // Set baud rate to 115200
  USART->BRR |= (22 << USART_BRR_DIV_Mantissa_Pos | 13 << USART_BRR_DIV_Fraction_Pos);
  USART->CR1 |= (USART_CR1_TE | USART_CR1_RE);
}

// Simple function to send characters over USART.
void sendChar(uint8_t data){
    USART->DR = (data & USART_DR_DR);
    while(!((USART->SR >> USART_SR_TC_Pos) & 1));
}
```

Ex. 3: Multiple tasks with two serial prints

```
#define USART USART2
#define UART_DELAY_MS 2000


// Strings to print from tasks.
const uint8_t str1[64] = "Hello from Task 1.\n";
const uint8_t str2[64] = "Hello from Task 2.\n";
```

```
// Task to print string over USART
static void printStringTask(void *pvParameters) {
  uint8_t * str = (uint8_t *) pvParameters;
  const TickType_t xDelay =
pdMS_TO_TICKS(UART_DELAY_MS);
  int i = 0;

  while(1) {
    do {
      sendChar(str[i]);
      i++;
    }
    while(str[i] != 0);

    i = 0;

    vTaskDelay(xDelay);
  }
}
```

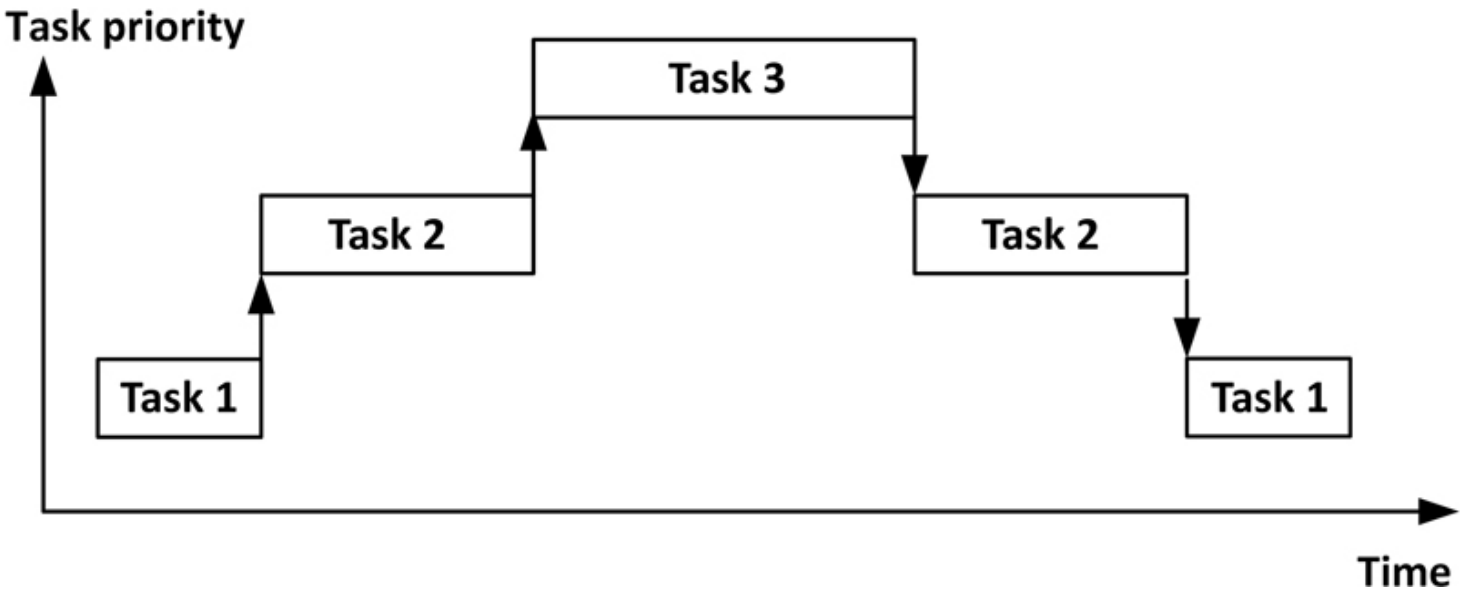
Ex. 3: Multiple tasks with two serial prints

- Which task prints first? How could you change this?
- Change duty cycle so that Task 1 prints once a second, Task 2 prints every other second.
  - How do you expect the tasks to execute now?

Preemptive scheduling

- Most common scheduling algorithm in real-time systems
- Tasks are assigned priorities
- Higher priority tasks can preempt lower priority tasks to take the CPU
- Need to be careful to assign priorities appropriately or you can starve lower priority tasks

Q: How are tasks and their priorities different than interrupts?



Ibrahim, Dogan. *ARM-Based Microcontroller Multitasking Projects: Using the FreeRTOS Multitasking Kernel*.

Ex. 4: Simple preemption example: poll button and blink LED and single print

```
// Task to poll button
static void pollButtonTask(void *pvParameters) {
  const TickType_t xDelay = pdMS_TO_TICKS(100); // Schedule every 100 ms
  volatile int i;

  while(1) {
    volatile int pin_val = 1;

    // Loop to check if the button is pressed (button is pulled low when pressed)
    // and blink LED rapidly while the button is pressed using a dummy loop.
    while(pin_val){
      pin_val = !((GPIOC->IDR >> BUTTON_PIN) & 1);
      if(pin_val) {
        toggleLED(LED_PIN);
        for(i=0; i < 400000; i++); // Dummy loop to do a delay.
      }
    }
    vTaskDelay(xDelay);
  }
}
```

Ex. 4: Simple preemption example: poll button and blink LED and single print

```
// Main function where initialization is performed and tasks are created.
int main()
{
    // Call initialization functions
    init_flash();   // Set up flash
    init_clock();   // Configure 84 MHz clock rate
    init_gpio();    // Initialize GPIO for LED
    init_uart();    // Initialize UART

    // Create tasks
    const size_t xRegTestStackSize = 250U; // Set value for stack for each task.
    xTaskCreate(toggleLedTask,      // Task function
                "Blink_1",          // Optional name for task
                xRegTestStackSize,  // Task stack size
                (void*)&led_1,      // void pointer to optional parameters
                1,                  // Task priority
                NULL);              // Handle to created task

    xTaskCreate(printStringTask, "Print_Test1", xRegTestStackSize, (void*)&str1, 2, NULL);
    xTaskCreate(pollButtonTask, "Poll_Button", xRegTestStackSize, NULL, 3, NULL);

    // Start the scheduler
    vTaskStartScheduler();

    // Infinite while loop. Should never get here unless the scheduler fails to start.
    while (1);
}
```

Ex. 4: Questions

- What will execution look like?
- Will the other tasks get any CPU time?
- What behavior do you expect to see if we change the priority of the print task such that it is higher than that of the poll button task?
- Task states
  - What happens if you press and release the button quickly? When does the print occur?
  - What happens if you hold the button for several seconds and then release it? When does the print occur then?
  - Why are these two cases different?
- How can we make this example more efficient?

Summary

- To understand the following key concepts of Real-time Operating Systems through examples in FreeRTOS
  - Task creation – tasks are like wrappers for C functions. They should never return and yield to the scheduler once they are done doing their work.
  - Basic scheduling – The scheduler decides what task should be running at any given time.
  - Task priorities and preemption – Task priorities help the scheduler decide between the importance of different tasks. Can be useful to distinguish between hard and soft deadlines and make sure they are met appropriately.

References

- Ibrahim, Dogan. *ARM-Based Microcontroller Multitasking Projects: Using the FreeRTOS Multitasking Kernel*. Netherlands, Elsevier Science, 2020.
- Barry, Richard. *Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide*. 2016.

Lecture Feedback

- What is the most important thing you learned in class today?
- What point was most unclear from lecture today?

https://forms.gle/Ay6MkpZ6x3xsW2Eb8