Course: CMPE244

Name: Sarika Satish Natu
ID: 014529138(EE)

Lab: #2
Date: 9/5/2020

Multiple Tasks

Scenario 1: task1 and task2 both running at low priority
Subcase 1: task1 is created before task2
Relevant code:

```c
static void task_one(void *task_parameter);
static void task_two(void *task_parameter);

int main(void) {
  create_blinky_tasks();
  create_uart_task();

  puts("Starting RTOS");
  xTaskCreate(task_one, "task1", configMINIMAL_STACK_SIZE, NULL, PRIORITY_LOW, NULL);
  xTaskCreate(task_two, "task2", configMINIMAL_STACK_SIZE, NULL, PRIORITY_LOW, NULL);
  vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs out of memory and fails

  return 0;
}

static void task_one(void *task_parameter) {
  while (true) {
    // Read existing main.c regarding when we should use fprintf(stderr...) in place of printf()
    // For this lab, we will use fprintf(stderr, ...)
    fprintf(stderr, "AAAAAAAAAAA");

    // Sleep for 100ms
    vTaskDelay(100);
  }
}

static void task_two(void *task_parameter) {
  while (true) {
    fprintf(stderr, "bbbbbbbbbbb");
    vTaskDelay(100);
  }
}
```

Telemetry output:

```
--------------------------------------------------------------------------------
peripherals_init(): Low level startup
WARNING: SD card could not be mounted

I2C slave detected at address: 0x38
I2C slave detected at address: 0x62
I2C slave detected at address: 0x72

entry_point(): Entering main()
Starting RTOS

List of commands (use help <name> to get full heblp if you see ..A.):
             crash : Deliberately crashes the system to demonstrate how ..b.
             A  i2c : i2c read 0xDD 0xRR <n>...
          tasAklist : Outputs list of RTOS tasks, CPU and stack usage....
-----------------------------------A----------------b-----------------------------
AAAbbbAAAAbbbbAbAAAAAbbbbAAAAbbbbAAAbbbbAAAAAAAAAbbbbAAAbbbbbbbbbAAAAAAAAAAAAbbbbbbbbb
AAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAbbb
bbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAA
bbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAA
AAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbb
AAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAbbb
bbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbb
```

Explanation of “How come 4(or 3 sometimes) characters are printed from each task? Why not 2 or 5, or 6?”
As the speed is defaulted to 38400bps and 10 bits of data used to send 1 byte, i.e. 3840 characters can be sent per second.
FreeRTOS tick rate is configured to be 1kHz, i.e. preemptive scheduling is occurring every 1ms repeatedly.
If 3840 characters can be sent in 1 sec, then how many characters can be sent in 1msec?
No of characters can be sent in 1 msec = 3840/1000 = 3.84 character.
Therefore, characters that can be printer can only be either of 4 or 3 of each task, but not 2 or 5 or 6.

Subcase 1: task2 is created before task1

```c
static void task_one(void *task_parameter);
static void task_two(void *task_parameter);

int main(void) {
  create_blinky_tasks();
  create_uart_task();

  puts("Starting RTOS");
  xTaskCreate(task_two, "task2", configMINIMAL_STACK_SIZE, NULL, PRIORITY_MEDIUM, NULL);
  xTaskCreate(task_one, "task1", configMINIMAL_STACK_SIZE, NULL, PRIORITY_MEDIUM, NULL);
  vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs out of memory and fails

  return 0;
}

static void task_one(void *task_parameter) {
  while (true) {
    // Read existing main.c regarding when we should use fprintf(stderr...) in place of printf()
    // For this lab, we will use fprintf(stderr, ...)
    fprintf(stderr, "AAAAAAAAAAAA");

    // Sleep for 100ms
    vTaskDelay(100);
  }
}

static void task_two(void *task_parameter) {
  while (true) {
    fprintf(stderr, "bbbbbbbbbbbb");
    vTaskDelay(100);
  }
}
```

```
--------------------------------------------------------------------------------
peripherals_init(): Low level startup
WARNING: SD card could not be mounted

I2C slave detected at address: 0x38
I2C slave detected at address: 0x62
I2C slave detected at address: 0x72

entry_point(): Entering main()
Starting RTOS

List of commands (use help <nameA> to get full help if you see ...):
          b   crash : DelibAerately crashes the system to demonstrate how ...
             b  i2c : i2c read 0xDD 0xRR <n>...
          tasklist : Outputs Alist of RTOS tasbks, CPU and stack usage....
-------------------A--------------------------------b----------------------------
AAbbbAAAAbbbbAbAAAAAbbbbAAAAbbbbAAAbbbbAAAAAAAAAbbbbAAAbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbb
AAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAA
bbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAA
bbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbb
AAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbb
AAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbb
```

Explanation:

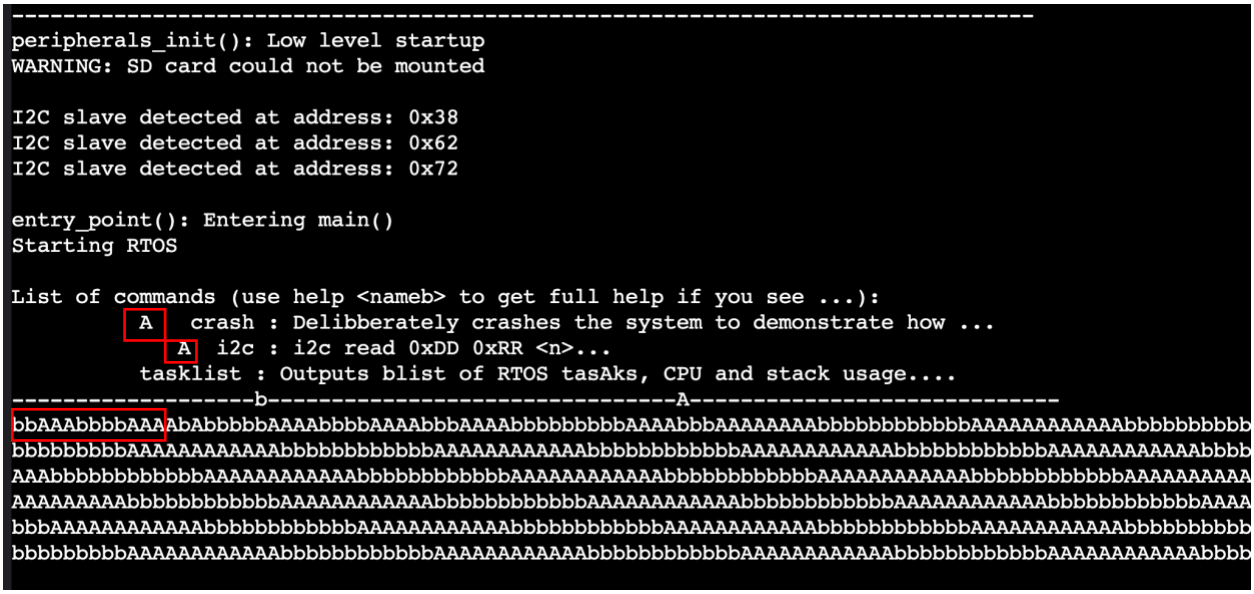
Even though the tasks are created in any order, the execution of the tasks depends on the scheduler and order of task execution cannot be predicted. The control over the order of task execution is out of scope and it completely depends on the scheduler.

Scenario 2: Same priority; Task 1 and Task 2 both running at medium priority

Relevant code:

```c
static void task_one(void *task_parameter);
static void task_two(void *task_parameter);

int main(void) {
  create_blinky_tasks();
  create_uart_task();

  puts("Starting RTOS");
  xTaskCreate(task_one, "task1", configMINIMAL_STACK_SIZE, NULL, PRIORITY_MEDIUM, NULL);
  xTaskCreate(task_two, "task2", configMINIMAL_STACK_SIZE, NULL, PRIORITY_MEDIUM, NULL);
  vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs out of memory and fails

  return 0;
}

static void task_one(void *task_parameter) {
  while (true) {
    // Read existing main.c regarding when we should use fprintf(stderr...) in place of printf()
    // For this lab, we will use fprintf(stderr, ...)
    fprintf(stderr, "AAAAAAAAAAA");

    // Sleep for 100ms
    vTaskDelay(100);
  }
}

static void task_two(void *task_parameter) {
  while (true) {
    fprintf(stderr, "bbbbbbbbbbb");
    vTaskDelay(100);
  }
}
```

Telemetry output:



Observation:
As both the tasks have same priority, task1 gets equal time of execution as task2 when task2 calls delay.
The order of the task execution completely depends on the scheduler. Therefore both the strings of task1 and task2 can be printed in any fashion, depending on which tasks does scheduler decide to execute.

Scenario 3: Task 1 has high priority and Task 2 has low priority

Relevant code:

```c
static void task_one(void *task_parameter);
static void task_two(void *task_parameter);

int main(void) {
  create_blinky_tasks();
  create_uart_task();

  puts("Starting RTOS");
  xTaskCreate(task_one, "task1", configMINIMAL_STACK_SIZE, NULL, PRIORITY_HIGH, NULL);
  xTaskCreate(task_two, "task2", configMINIMAL_STACK_SIZE, NULL, PRIORITY_LOW, NULL);
  vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs out of memory and fails

  return 0;
}

static void task_one(void *task_parameter) {
  while (true) {
    // Read existing main.c regarding when we should use fprintf(stderr...) in place of printf()
    // For this lab, we will use fprintf(stderr, ...)
    fprintf(stderr, "AAAAAAAAAAA");

    // Sleep for 100ms
    vTaskDelay(100);
  }
}

static void task_two(void *task_parameter) {
  while (true) {
    fprintf(stderr, "bbbbbbbbbbb");
    vTaskDelay(100);
  }
}
```

Telemetry output:

```
--------------------------------------------------------------------------------
peripherals_init(): Low level startup
WARNING: SD card could not be mounted

I2C slave detected at address: 0x38
I2C slave detected at address: 0x62
I2C slave detected at address: 0x72

entry_point(): Entering main()
Starting RTOS
AAAAAAAAAAAAbbbbbbbbbbb
List of commands (use help <name> to get full help if you see ...):
             crash : Deliberately crashes the system to demonstrate how ...
               i2c : i2c read 0xDD 0xRR <n>...
          tasklist : Outputs list of RTOS tasks, CPU and stack usage....
--------------------------------------------------------------------------------
AAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAA
bbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAA
bbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbb
AAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbb
AAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAA
bbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAA
```

Explanation:
Here task1 has high priority and task2 has low priority.

Therefore, in output screen, it can be seen that task1 gets executed first printing ‘A’ sequence later followed by task2 printing ‘b’ sequence.

Scenario 4: Task 1 has low priority and Task 2 has high priority

Relevant code:

```
static void task_one(void *task_parameter);
static void task_two(void *task_parameter);

int main(void) {
  create_blinky_tasks();
  create_uart_task();

  puts("Starting RTOS");
  xTaskCreate(task_one, "task1", configMINIMAL_STACK_SIZE, NULL, PRIORITY_LOW, NULL);
  xTaskCreate(task_two, "task2", configMINIMAL_STACK_SIZE, NULL, PRIORITY_HIGH, NULL);
  vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs out of memory and fails

  return 0;
}

static void task_one(void *task_parameter) {
  while (true) {
    // Read existing main.c regarding when we should use fprintf(stderr...) in place of printf()
    // For this lab, we will use fprintf(stderr, ...)
    fprintf(stderr, "AAAAAAAAAAA");

    // Sleep for 100ms
    vTaskDelay(100);
  }
}

static void task_two(void *task_parameter) {
  while (true) {
    fprintf(stderr, "bbbbbbbbbbb");
    vTaskDelay(100);
  }
}
```

Telemetry output:

```
--------------------------------------------------------------------------------
peripherals_init(): Low level startup
WARNING: SD card could not be mounted

I2C slave detected at address: 0x38
I2C slave detected at address: 0x62
I2C slave detected at address: 0x72

entry_point(): Entering main()
Starting RTOS
bbbbbbbbbbbbAAAAAAAAAAAA
List of commands (use help <name> to get full help if you see ...):
             crash : Deliberately crashes the system to demonstrate how ...
               i2c : i2c read 0xDD 0xRR <n>...
          tasklist : Outputs list of RTOS tasks, CPU and stack usage....
--------------------------------------------------------------------------------
bbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAA
AAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAA
AAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbb
bbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbbAAAAAAAAAAAAbbbb
```

Explanation:
Here task2 has high priority and task2 has low priority.
Therefore, in output screen, it can be seen that task2 gets executed first printing ‘b’ sequence later followed by task1 printing ‘A’ sequence.