Course: CMPE244

Name: Sarika Satish Natu
ID: 014529138(EE)

Lab: #2
Date: 9/7/2020

Multiple Tasks

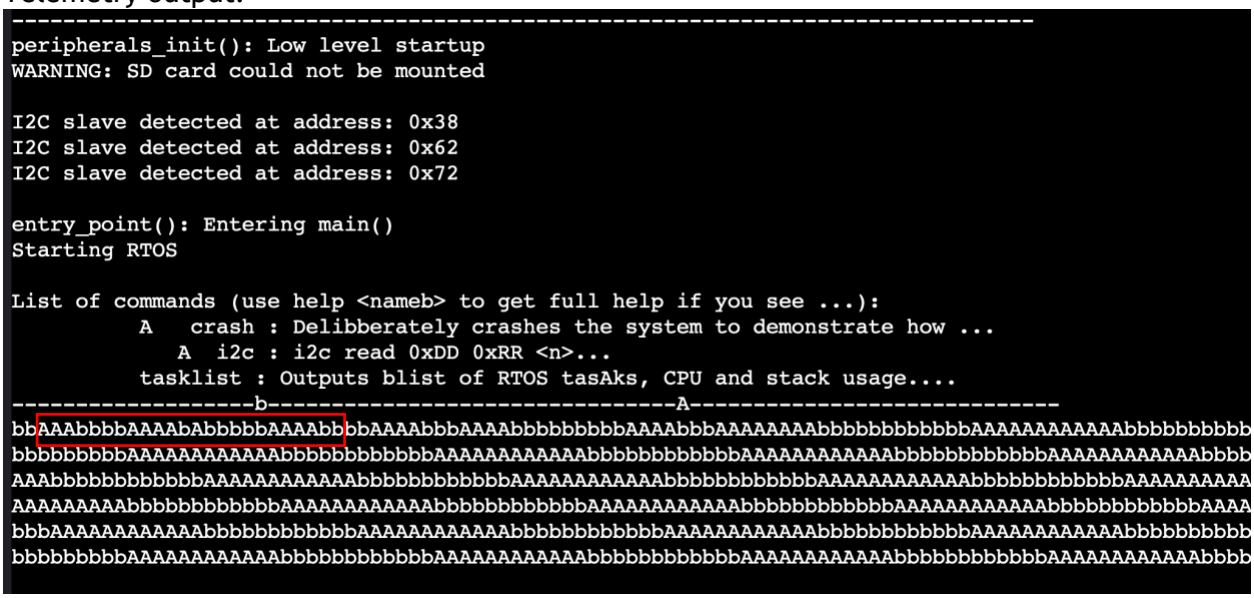
Scenario 1: task1 and task2 'xTaskCreate' arguments setup.

Subcase 1: task1 and task2 running at same 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_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:



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 ≅ 3 or 4 characters
Therefore, characters that can be printer can only be either of 4 or 3 of each task, but not 2 or 5 or 6.
This is known as time slicing performed by the scheduler in round robin fashion.

Subcase 2: task1 and task2 running at different priorities.

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....
--------------------------------------------------------------------------------
AAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAA
bbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAA
bbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbb
AAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbb
AAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAA
bbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAAbbbbbbbbbbbbAAA
```

Explanation of “Alter the priority of one of the tasks and note down the observations. Note down WHAT you see and WHY.”

In this case where one task has higher priority over the second task, the higher priority task performs its task operation and preempts low priority task from executing. Thus, low priority performs its operations after completion of high priority task. Therefore, in this scenario, complete series of ‘A’s is printed followed by complete series of ‘b’s or vice versa depending on which task has higher priority, but not in 4 or 3 character pattern as in earlier case.

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

```
--------------------------------------------------------------------------------
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 <nameb> to get full help if you see ...):
         A   crash : Delibberately crashes the system to demonstrate how ...
            A  i2c : i2c read 0xDD 0xRR <n>...
          tasklist : Outputs blist of RTOS tasAks, CPU and stack usage....
-------------------b-------------------------------A----------------------------
bbAAAbbbbAAAAbAbbbbbbAAAAbbbbAAAAbbbAAAAbbbbbbbbbbAAAAbbbAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbb
bbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbb
AAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAA
AAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAA
bbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbb
bbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbbAAAAAAAAAAAAbbbb
```

Observation:
As both the tasks have same priority, task1 gets equal time of execution as task2 when the other task goes into 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 pattern, depending on which tasks does scheduler decide to execute and depending on UART speed.

Consider if task2 is created before task 1.
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_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, "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 <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--------------------------
AAbbbAAAAbbbbAbAAAAAbbbbAAAAbbbbAAAbbbbAAAAAAAAAbbbbAAAbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbb
AAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAA
bbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAA
bbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbb
AAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbb
AAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbbAAAAAAAAAAAAbbbbbbbbbbbb
```

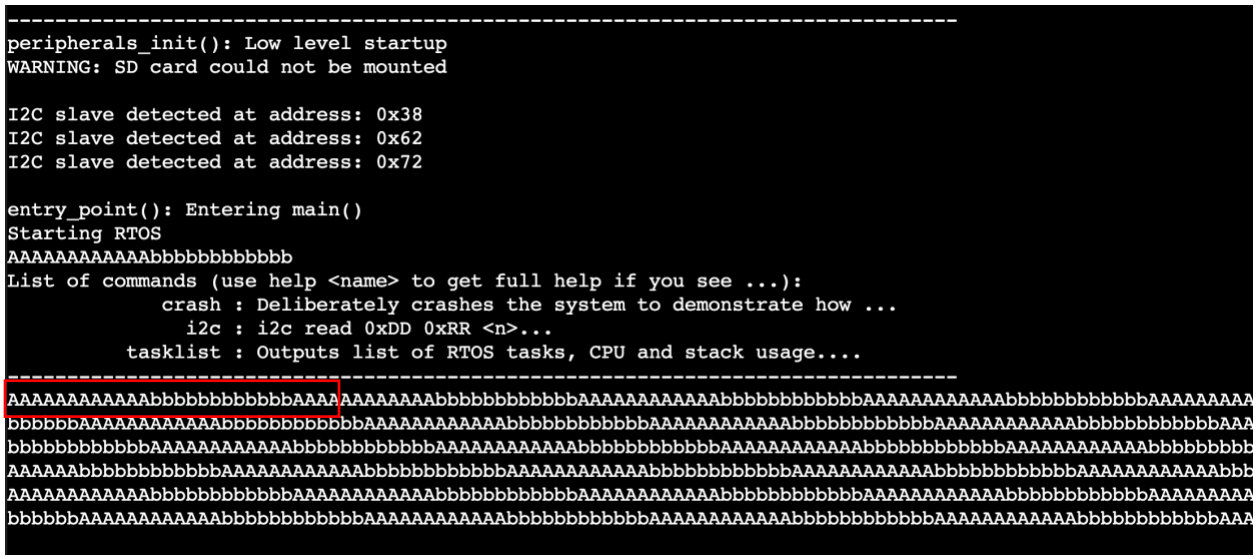
Even though the tasks are created in reverse order, the execution of the tasks depends on the scheduler and order of task execution cannot be predicted. The control over the order of task execution is out of scope and it completely depends on the scheduler. This is called as time slicing performed by the round robin scheduler.

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



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

Scenario 4: Task 1 has low priority and Task 2 has high 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_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
bbbbbbbbbbbbAAAAAAAAAAA
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.

This is the opposite case of the earlier case. So here on the output screen, it can be seen that task2 gets executed first printing ‘b’s sequence later followed by task1 printing ‘A’s sequence, as task2 preempts task1 from executing.