FreeRTOS Task (C)

Nick Scheele

main.c

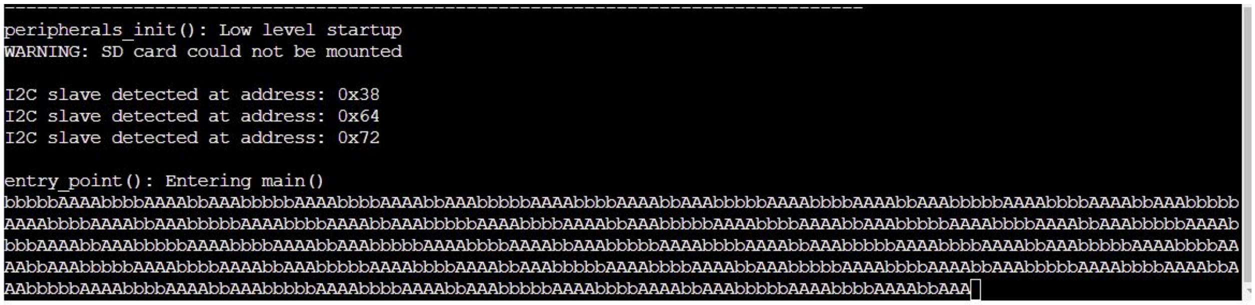
```c
#define outOfTheBox 0

int main(void) { // main function for project
#if outOfTheBox

 create_blinky_tasks();
 create_uart_task();

 puts("Starting RTOS");

#else
 /**
  * Observe and explain the following scenarios:
  *
  * 1) Same Priority:      task_one = 1, task_two = 1
  * 2) Different Priority: task_one = 2, task_two = 1
  * 3) Different Priority: task_one = 1, task_two = 2
  *
  * Note: Priority levels are defined at FreeRTOSConfig.h
  * Higher number = higher priority
  *
  * Turn in screen shots of what you observed
  * as well as an explanation of what you observed
  */
 xTaskCreate(task_one, /*description*/ "task_one", /*stack depth*/ 4096 /
sizeof(void *), /*parameter*/ (void *)1,
             /*priority*/ 1, /*optional handle*/ NULL);
 xTaskCreate(task_two, /*description*/ "task_two", /*stack depth*/ 4096 /
sizeof(void *), /*parameter*/ (void *)1,
             /*priority*/ 2, /*optional handle*/ NULL);
#endif

 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) {
   fprintf(stderr, "AAAAAAAAAAA");

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

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

   vTaskDelay(100); // sleep for 100ms
 }
}
```

peripherals_init.c

```
static void peripherals_init__uart0_init(void) {
 // Do not do any bufferring for standard input otherwise getchar(), scanf() may not work
 setvbuf(stdin, 0, _IONBF, 0);

 // Note: PIN functions are initialized by board_io__initialize() for P0.2(Tx) and P0.3(Rx)
 uart__init(UART__0, clock__get_peripheral_clock_hz(), 38400); // Chagned 115200 to 38400

 // You can use xQueueCreate() that uses malloc() as it is an easier API to work with, however, we opt to
 // use xQueueCreateStatic() to provide reference on how to create RTOS queue without dynamic memory allocation

 // Memory for the queue data structure
 static StaticQueue_t rxq_struct;
 static StaticQueue_t txq_struct;

 // Memory where the queue actually stores the data
 static uint8_t rxq_storage[32];
 static uint8_t txq_storage[128];

 // Make UART more efficient by backing it with RTOS queues (optional but highly recommended with RTOS)
 QueueHandle_t rxq_handle = xQueueCreateStatic(sizeof(rxq_storage), sizeof(char), rxq_storage, &rxq_struct);
 QueueHandle_t txq_handle = xQueueCreateStatic(sizeof(txq_storage), sizeof(char), txq_storage, &txq_struct);

 uart__enable_queues(UART__0, txq_handle, rxq_handle);
}
```

How come 4(or 3 sometimes) characters are printed from each task? Why not 2 or 5, or 6?

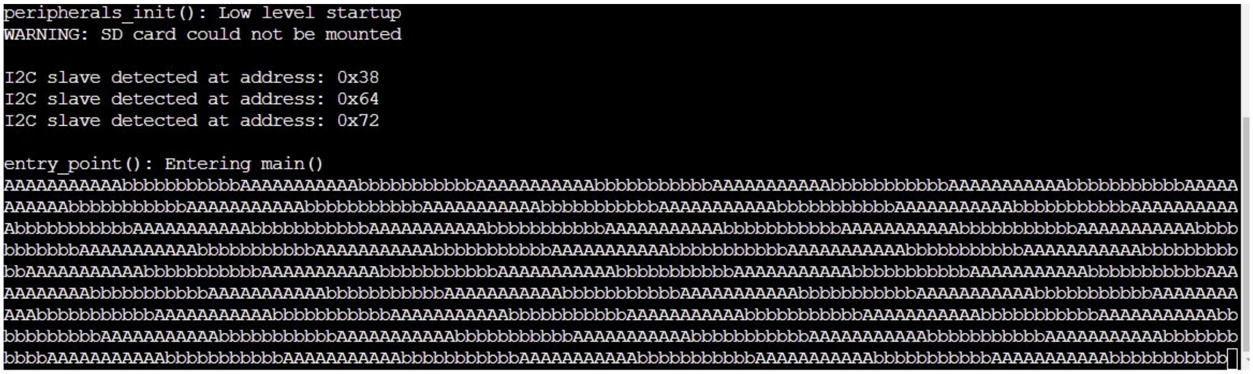
3840 characters/s is equivalent to 3.8 characters/ms and with 1ms scheduling, the result is 3-4 characters per scheduled run of each task in the round robin.



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

I2C slave detected at address: 0x38
I2C slave detected at address: 0x64
I2C slave detected at address: 0x72

entry_point(): Entering main()
bbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbb
AAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAb
bbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAA
AAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbA
AAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAAbbbbbAAAAbbbbAAAAbbAAA
```

Figure 1: equal priority

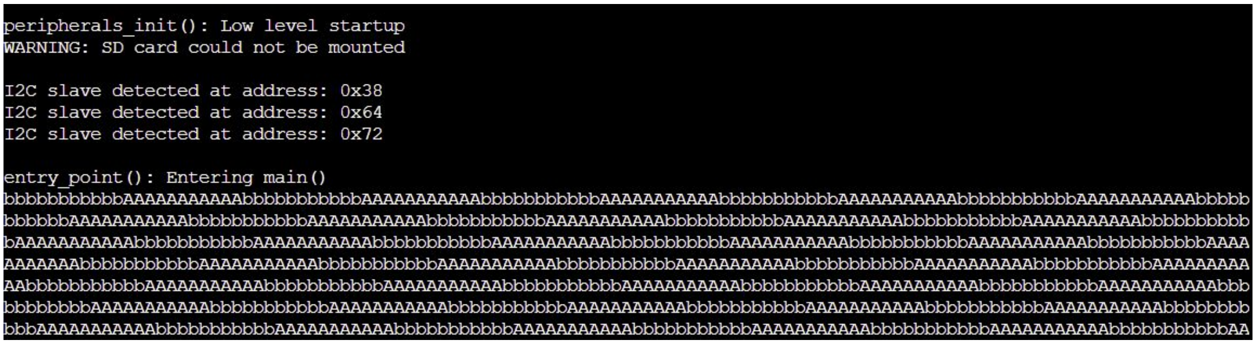
Neither task runs to completion because the scheduler needs to split time equally between the two tasks.



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

I2C slave detected at address: 0x38
I2C slave detected at address: 0x64
I2C slave detected at address: 0x72

entry_point(): Entering main()
AAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAA
AAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAA
AbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbb
bbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbb
bbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAA
AAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAA
AAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbb
bbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbb
bbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbb
```

Figure 2: Task_one priority

Task_one has priority in this case, therefore all the letters “A” are printed and when Task_one is sleeping Task_two runs.



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

I2C slave detected at address: 0x38
I2C slave detected at address: 0x64
I2C slave detected at address: 0x72

entry_point(): Entering main()
bbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbb
bbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbb
bAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAA
AAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAA
AAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbb
bbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbb
bbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAAAAAAAAAAAbbbbbbbbbbbAA
```

Figure 3: Task_two priority

Task_two has priority in this case, therefore all the letters “b” are printed and when Task_two is sleeping Task_one runs.