

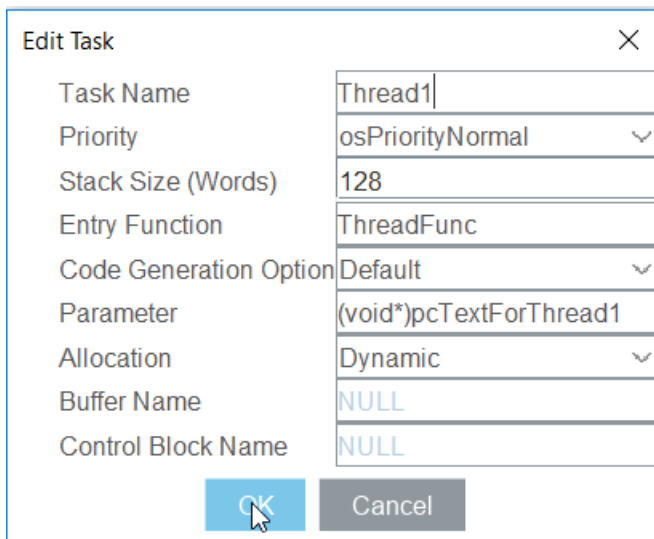
Real-time Embedded Systems Lab Manual

Thread Management in FreeRTOS – Part 2 (Lab 2 - 3)

Lab 2: Using the thread parameter

The two threads functions *Thread1Func* and *Thread2Func* defined in lab 1 are almost identical, except the text string they print out. We will create a single Thread function implementation and use two instances of this same thread function to remove the duplication. The parameter *argument* of a thread function is used to pass into each thread instance the string that it should print out.

1. Follow the steps 1 - 4 in the getting started lab to create a new project lab2, to configure the MCU.
2. Select Middleware >> FREERTOS and edit the two threads in the FREERTOS configuration panel. Modify the **Entry Function** of both Thread1 and Thread2 as “ThreadFunc”, and modify the **Parameter** as “(void*)pcTextForThread1” for Thread1 and the **Parameter** as “(void*)pcTextForThread2” for Thread2.



| Edit Task | |
|------------------------|-------------------------|
| Task Name | Thread1 |
| Priority | osPriorityNormal |
| Stack Size (Words) | 128 |
| Entry Function | ThreadFunc |
| Code Generation Option | Default |
| Parameter | (void*)pcTextForThread1 |
| Allocation | Dynamic |
| Buffer Name | NULL |
| Control Block Name | NULL |
| <div>OK Cancel</div> | |

| | |
|------------------------|-------------------------|
| Task Name | Thread2 |
| Priority | osPriorityNormal |
| Stack Size (Words) | 128 |
| Entry Function | ThreadFunc |
| Code Generation Option | Default |
| Parameter | (void*)pcTextForThread2 |
| Allocation | Dynamic |
| Buffer Name | NULL |
| Control Block Name | NULL |

OK Cancel

3. Edit the User Constants by adding two character string constants: constant Name as **pcTextForThread1** and constant Value as **"\n\rThread 1 is running\n\r"**; another constant Name as **pcTextForThread2** and constant Value as **"\n\rThread 2 is running\n\r"**. The constant with the name **mainDELAY_LOOP_COUNT** and the Value **0xffff** used in lab 1 is also used here.

| | |
|----------------|-------------------------------|
| constant Name | pcTextForThread1 |
| constant Value | "\n\rThread 1 is running\n\r" |

OK Cancel

| | |
|----------------|-------------------------------|
| constant Name | pcTextForThread2 |
| constant Value | "\n\rThread 2 is running\n\r" |

OK Cancel

Together, we define three constants.

| | | |
|-------------------------|----------------------|-----------------------|
| ✓ Timers and Semaphores | ✓ Mutexes | ✓ FreeRTOS Heap Usage |
| ✓ User Constants | ✓ Tasks and Queues | |
| ✓ Config parameters | ✓ Include parameters | |

Search Constants

| Constant Name | Constant Value |
|----------------------|-------------------------------|
| mainDELAY_LOOP_COUNT | 0xfffff |
| pcTextForThread1 | "\n\rThread 1 is running\n\r" |
| pcTextForThread2 | "\n\rThread 2 is running\n\r" |

4. Save the configuration file to generate initialization code.
5. Edit main.c similar to that in lab1 to define `putchUSART1()` and `putsUSART1()`.

```

75  /* USER CODE BEGIN 0 */
76  void putchUSART1 (char ch)
77  {
78      /* Place your implementation of fputc here */
79      /* e.g. write a character to the serial port and Loop until the end of transmission */
80      while (HAL_OK != HAL_UART_Transmit(&huart1, (uint8_t *) &ch, 1, 30000))
81      {
82      };
83      }
84  }
85
86  void putsUSART1 (char* ptr)
87  {
88      while(*ptr)
89      {
90          putchUSART1(*ptr++);
91      }
92  }
93  /* USER CODE END 0 */
..

122  /* Initialize all configured peripherals */
123  MX_GPIO_Init();
124  MX_USART1_UART_Init();
125  /* USER CODE BEGIN 2 */
126  putsUSART1("\n\rFreeRTOS Lab 2\n\r");
127  /* USER CODE END 2 */

```

6. Modify the definition of the thread function *ThreadFunc* in *main.c*, Remember casting the type of the function parameter *argument* from *void ** to *char **. You could use this code as a reference.

```
514 /* USER CODE END Header_ThreadFunc */
515 void ThreadFunc(void *argument)
516 {
517     /* USER CODE BEGIN 5 */
518     volatile unsigned long ul;
519     /* Infinite loop */
520     for(;;)
521     {
522         putsUSART1((char*)argument);
523
524         for(ul = 0; ul < mainDELAY_LOOP_COUNT; ul++)
525             {}
526
527         //osDelay(1);
528     }
529     /* USER CODE END 5 */
530 }
```

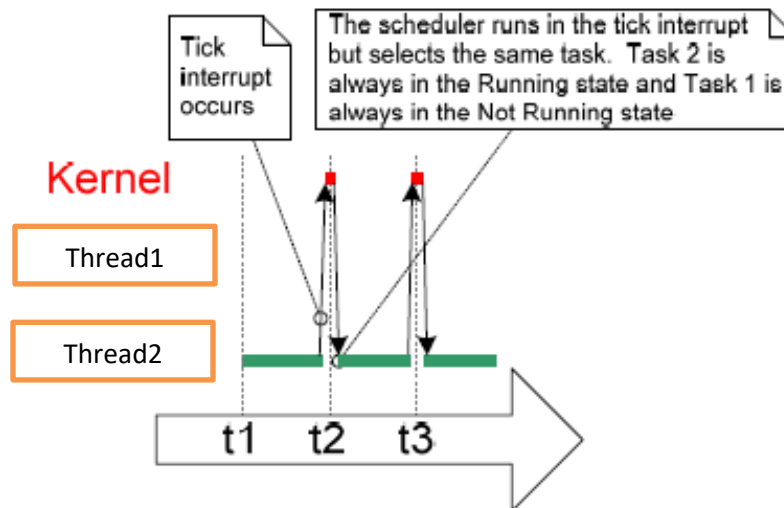
7. Build and debug the project as lab1.
8. Open the tera-term to check the print messages (the output should be same as lab 1).

Lab 3: Experimenting with priorities

In the Lab exercise 1 and 2, two threads have been created at the same priority, so both entered and exited the Running state in turn. This lab looks at what happens when we change the priority of one of the two threads created in Lab2.

- In the main() function, assign the priority of Thread1 as *osPriorityNormal*, and the priority of the Thread2 as *osPriorityNormal1*.
The function *ThreadFunc()* that implements both threads remain the same: it still simply prints out a string periodically using a null *for* loop to create a delay.
- Rebuild and debug the project.
- Open the tera term to check the print messages, compare the differences from lab 2.

The following figure illustrates the expected execution sequence. Thread1 with lower priority has no chance to run.



Questions:

1. How many CMSIS-RTOSv2 API functions are called in the main.c file? Please list them.
2. Set a breakpoint at one of the *osThreadNew()* statement in the main(). Step into this function and check which FreeRTOS API function(s) are called?

Please give your answers in the lab report.