Real-time Embedded Systems Lab Manual

Resource Management in FreeRTOS (lab15 - 16)

Lab 15 Rewriting vPrintString() to use a semaphore

This example creates a new version of *vPrintString()* called *prvNewPrintString()*, then calls the new function from multiple tasks. *prvNewPrintString()* is functionally identical to *vPrintString()*, but uses a mutex to control access to standard out in place of the basic crtical section.

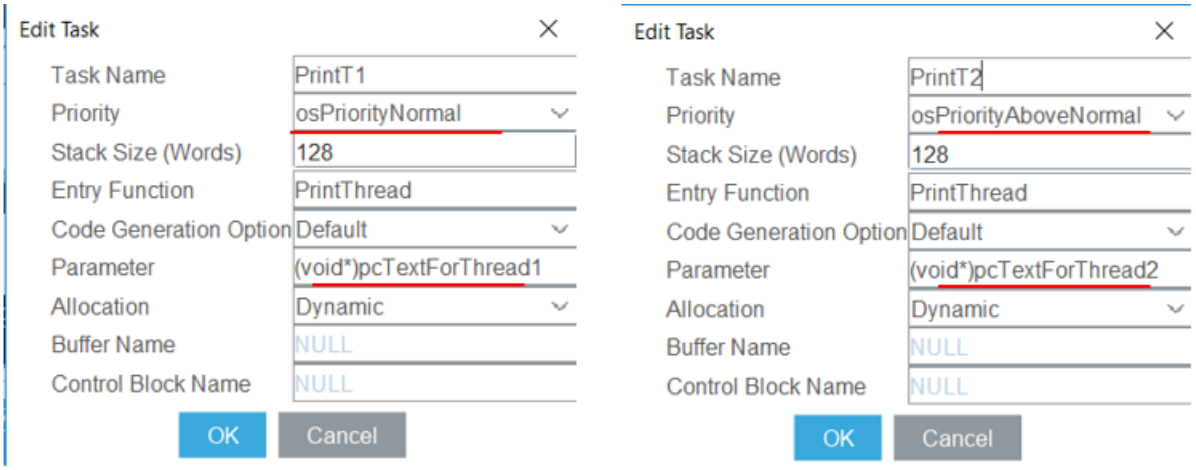
prvNewPrintString() is called repeatedly by two instances of a thread implemented by *PrintThread* (). A random delay time is used between each call. The thread paramter is used to pass a unique string into each instance of the thread.

In the *main()* function, we create the mutex, two threads, and then starts the scheduler. The two instances of *PrintThread* () are created at different priorities, so the lower priority thread will sometimes be pre-empted by the higher priority thread. A mutex is used to ensure each thread gets mutally exclusive access to the USART1 channel, even when pre-emption occurs, the strings that are displayed will be corrected and in no way corrupted. The frequency of the pre-emption can be increased by reducing the maximum time the threads spend in the Blocked state, which is defaulted to 0x1ff ticks.

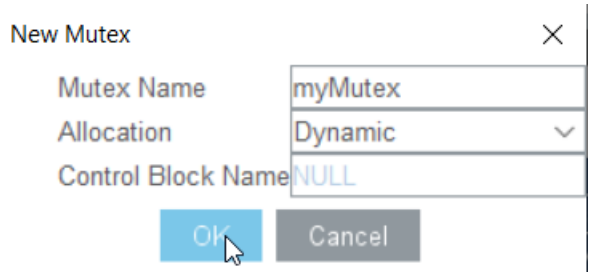
1. Assume that we use the MCU configuration file of lab12 as a start point and copy it to a new directory for lab15. You could also use the configuration file of any previous lab. Just make sure the following modes are set:
   a) The debug **Mode** of the **SYS** module under **System Core** category is set **Serial Wire**; the Timebase Source is set as TIM6 or TIM7.
   b) The **Mode** of the **USART1** module under the **Connectivity** category is set **Asynchronous;**
   c) The **interface** of the **FreeRTOS** under **Middleware** category is set **CMSIS_V2**.

   If you did not save the cubeMX configuration file, please follow the step 1 – 4 in the first lab manual *ThreadLab_p1_CMSISv2.docx*.

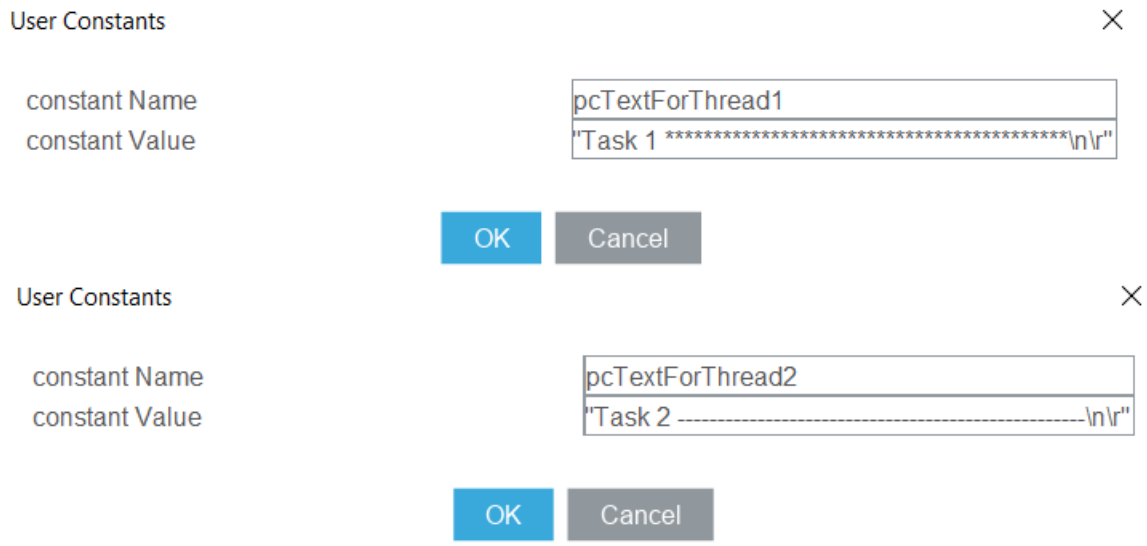
2. Select Middleware>>FREERTOS, open the "Tasks and Queues" tab on the FREERTOS Configuration panel. Define two periodic thread instances: *PrintT1* and *PrintT2* from the same function *PrintThread*, but with different **Priority** and **Parameter**.



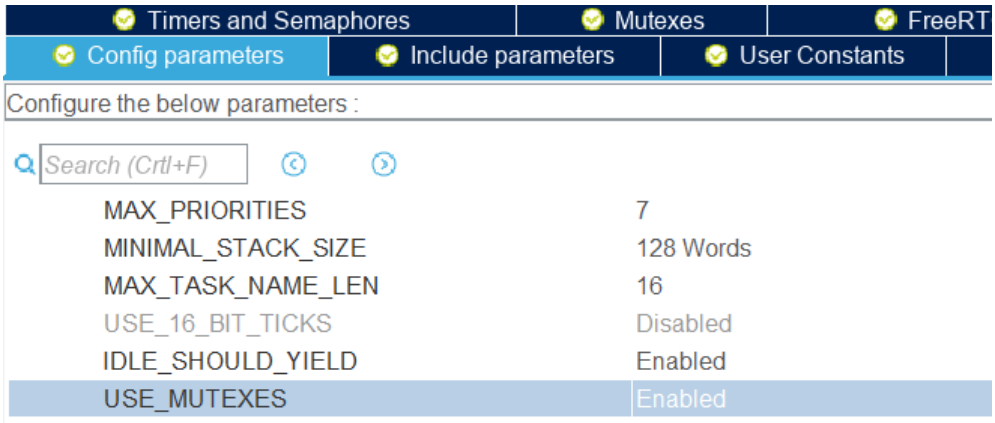
3. Open the “Mutexes” tab on the FREERTOS Configuration panel. Define one mutex “myMutex”.



4. Open the “User Constants” tab on the FREERTOS Configuration panel. Define two string constants with a **long** list of characters.
   pcTextForThread1: **"Task 1 ********** \n\r"**
   pcTextForThread2: "Task 2 ----------\n\r"



5. Open the “Config parameters” tab on the FREERTOS Configuration panel. Make sure “USE_MUTEXES” be “enabled”.



6. Save the configuration file to generate code.
7. Edit the main.c file.
   a) Define putchUSART1() and putsUSART1() to use the USART1 channel as previous labs.

```
/* USER CODE BEGIN 0 */
void putchUSART1 (char ch)
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the serial port and Loop until the end of transmission */
  while (HAL_OK != HAL_UART_Transmit(&huart1, (uint8_t *) &ch, 1, 30000))
  {
    ;
  }
}

void putsUSART1 (char* ptr)
{
    while(*ptr)
    {
        putchUSART1(*ptr++);
    }
}
/* USER CODE END 0 */
```

   b) As we will use the standard library function *rand()* to generate a random value in this project, include the corresponding header file “stdlib.h”.

```
/* USER CODE BEGIN Includes */
#include <stdlib.h>
/* USER CODE END Includes */
```

   c) Edit the main() function by calling putsUSART1() as below.

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  putsUSART1("\n\rFreeRTOS Lab 15\n\r");
  /* USER CODE END 2 */
```

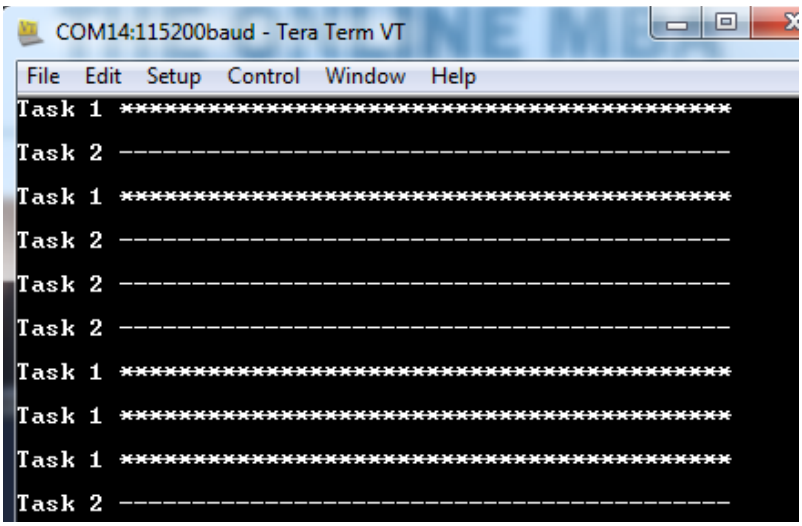
c) Define the prvNewPrintString() function to safely print messages using mutex.

```
/* USER CODE BEGIN 4 */
static void prvNewPrintString(const char *pcString)
{
    osMutexWait(myMutexHandle, osWaitForever);
    {
        /*THe following line will only execute once the mutex has been
         * successfully obtained - so the USART1 channel can be
         * accessed freely. */
        putsUSART1(pcString);
    }
    osMutexRelease(myMutexHandle);
}
/* USER CODE END 4 */
```

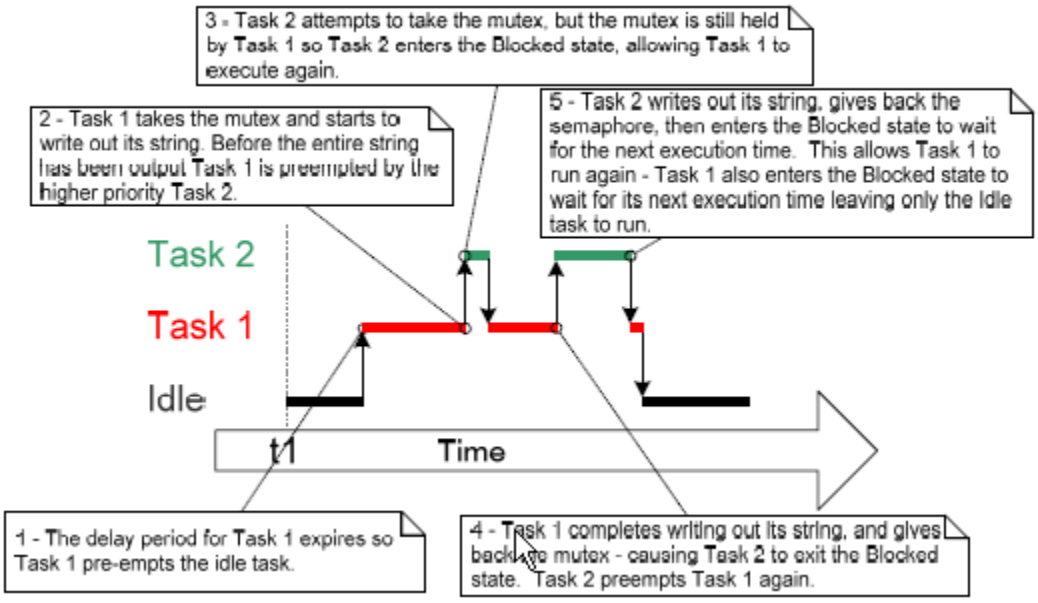
Define the printThread() function.

```
/* USER CODE END Header_PrintThread */
void PrintThread(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
      prvNewPrintString((char *)argument);
      osDelay((rand() & 0x1FF));
  }
  /* USER CODE END 5 */
}
```

The output produced when Example 15 is executed is shown below.



A possible execution seuqnece is described in the following.



Lab 16 Rewriting vPrintString() to use a gatekeeper task

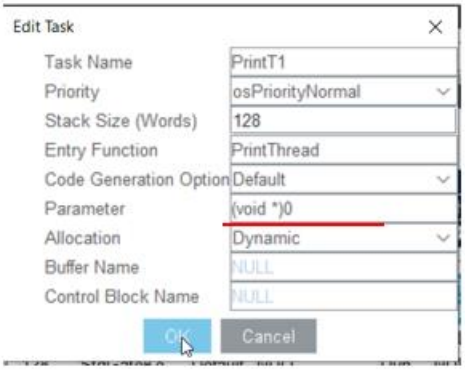
This example provides an alternative implementation for vPrintString(). This time, a gatekeeper thread is used to manage access to the USART1. When a thread wants to write a message to the tera terminal via USART, it does not call a print function directly but, instead, sends the message to the gatekeeper.

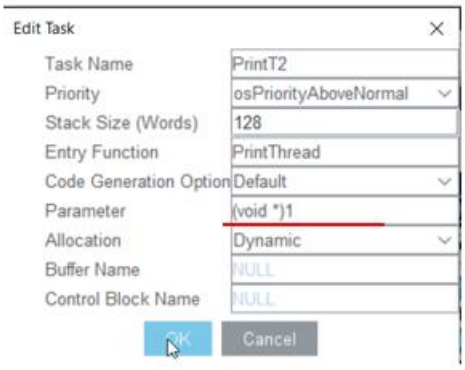
The gatekeeper thread uses an osMailQ to serialize access to the USART. The internal implementation of the thread does not have to consider mutual exclusion because it is the only thread permitted to access the USART directly.

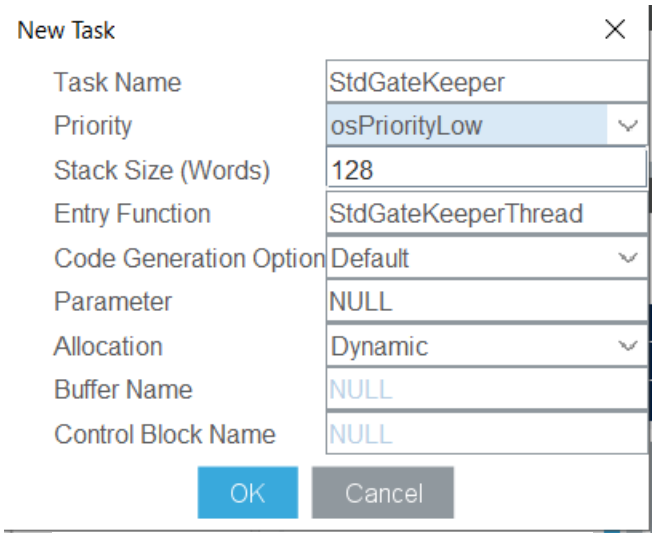
The gatekeeper thread spends most of its time in the blocked state, waiting for messages to arrive on the Mail. When a message arrives, the gatekeeper writes the message to the USART, before returning to the Blocked state to wait for the next message.

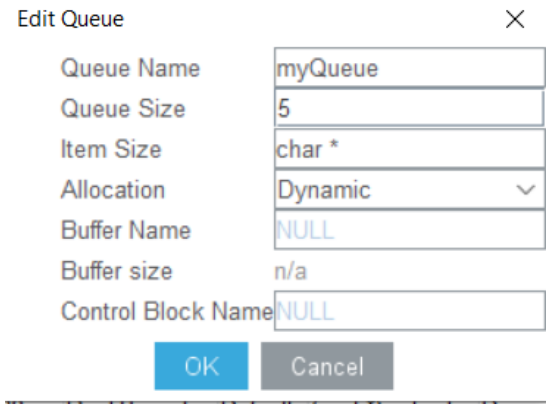
Create a new folder for lab 16, and configure peripherals as lab15, (You could copy *lab15.ioc* and rename it as *lab16.ioc* in the newly created folder as the starting point.)

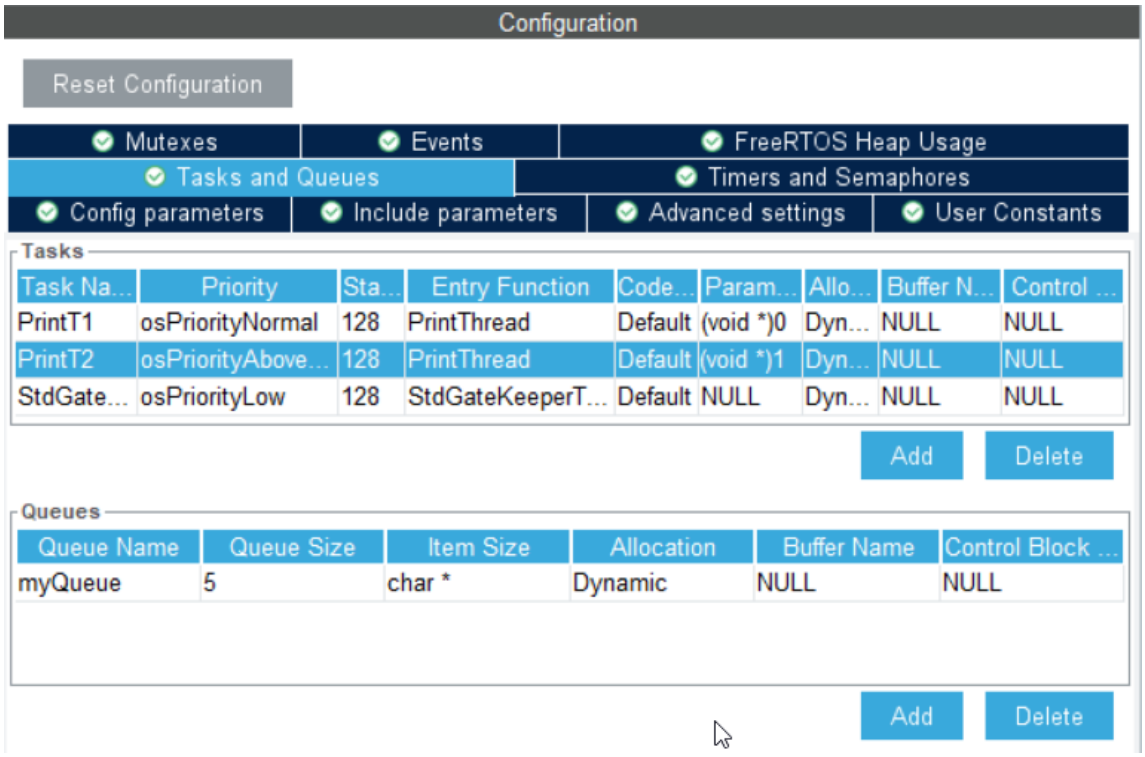
1. Select Middleware>>FREERTOS, open the "Tasks and Queues" tab on the FREERTOS Configuration panel. Revise the **Parameters** of two periodic threads in lab15 as below. Create the third thread with the lowest priority (i.e., osPriorityIdle) and a queue with the *char* * Item type and the queue size 5. This is the only task that is permitted to access standard out.



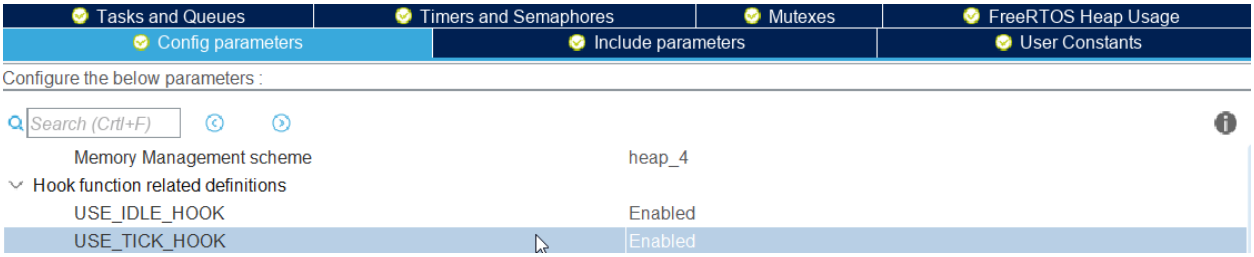



New Task
Task Name
StdGateKeeper
Priority
osPriorityLow
Stack Size (Words)
128
Entry Function
StdGateKeeperThread
Code Generation Option
Default
Parameter
NULL
Allocation
Dynamic
Buffer Name
NULL
Control Block Name
NULL
OK
Cancel


Edit Queue
Queue Name
myQueue
Queue Size
5
Item Size
char *
Allocation
Dynamic
Buffer Name
NULL
Buffer size
n/a
Control Block Name
NULL
OK
Cancel


Configuration
Reset Configuration
Mutexes
Events
FreeRTOS Heap Usage
Tasks and Queues
Timers and Semaphores
Config parameters
Include parameters
Advanced settings
User Constants
Tasks
Task Na...
Priority
Sta...
Entry Function
Code...
Param...
Allo...
Buffer N...
Control ...
PrintT1
osPriorityNormal
128
PrintThread
Default
(void *)0
Dyn...
NULL
NULL
PrintT2
osPriorityAbove...
128
PrintThread
Default
(void *)1
Dyn...
NULL
NULL
StdGate...
osPriorityLow
128
StdGateKeeperT...
Default
NULL
Dyn...
NULL
NULL
Add
Delete
Queues
Queue Name
Queue Size
Item Size
Allocation
Buffer Name
Control Block ...
myQueue
5
char *
Dynamic
NULL
NULL
Add
Delete

2. Open the “Mutexes” tab on the FREERTOS Configuration panel. Remove the previously created mutex in lab 15 as we will not need mutex to protect the USART channel.
3. Open the “User Constants” tab on the FREERTOS Configuration panel. Remove two string constants as we will declare them in an array type variable of the main.c file.
4. Open the “Config parameters” tab on the FREERTOS Configuration panel. Make sure “USE_TICK_HOOK” be “enabled”.



5. Save the configuration file to generate the code.
6. Edit the main.h file.
   a) Within /* USER CODE BEGIN EFP */ and /* USER CODE END EFP */, declare a function prototype *vApplicationTickHookFunc(void)*. It will be later called in the *void vApplicationTickHook(void)*.

```
55 /* USER CODE BEGIN EFP */
56 void vApplicationTickHookFunc(void);
57 /* USER CODE END EFP */
```

7. Edit the main.c file.
   a) Define putchUSART1() and putsUSART1() to use the USART1 channel as lab 15.

```
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 */
```

   b) Define a string array.

```
/* USER CODE BEGIN PV */
static char *pcStringsToPrint[] =
{
  "\n\rTask 1 ****************************************\n\r",
  "\n\rTask 2 ----------------------------------------\n\r",
  "\n\rMessage printed from tick hook interrupt \r\n"
};
/* USER CODE END PV */
```

c) In the main() function, add the following statements, and keep the generated code unchanged.

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  putsUSART1("\n\rFreeRTOS Lab 16\n\r");
  /* USER CODE END 2 */
```

d) Implement the *vApplicationTickHook()* function.

- Declare a static integer counter (i.e., named as *iCount*) and initialize it to 0;
- Increment the counter every Tick period (i.e., 1ms);
- When the counter reaches 200, send a specific string message to the Queue and reset the counter to zero. (You could set the max count a larger value than 200 and try.)

```
/* USER CODE BEGIN 4 */
void vApplicationTickHookFunc(void)
{
  static int iCount = 0;
  iCount++;
  if(iCount > 200)
  {
    osMessageQueuePut(myQueueHandle, &(pcStringsToPrint[2]), 0, 0);
    iCount = 0;
  }
}
/* USER CODE END 4 */
```

e) Define the PrintThread() thread function.

- Use input argument as index to access the global array of strings variable *pcStringsToPrint* and send selected string to the message queue.
- Then use *rand()* library function to generate random period delay. This actually implements the behavior of an aperiodic task.

```
553  /* USER CODE END Header_PrintThread */
554  void PrintThread(void *argument)
555  {
556    /* USER CODE BEGIN 5 */
557    /* Infinite loop */
558    for(;;)
559    {
560      osMessageQueuePut(myQueueHandle, &(pcStringsToPrint[(int)argument]), 0, 0);
561      osDelay(rand() & 0x1FF); // Use rand() to generate aperiodic task
562    }
563    /* USER CODE END 5 */
564  }
```

f) Define the StdGatekeeperThread() thread function.

```
585 /* USER CODE END Header_StdGateKeeperThread */
586 void StdGateKeeperThread(void *argument)
587 {
588   /* USER CODE BEGIN StdGateKeeperThread */
589     char *stringToPrint;
590   /* Infinite loop */
591   for(;;)
592   {
593       osMessageQueueGet(myQueueHandle, &(stringToPrint), 0, osWaitForever);
594       putsUSART1(stringToPrint);
595     //osDelay(1);
596   }
597   /* USER CODE END StdGateKeeperThread */
598 }
```

The output should be like the following.

```
FreeRTOS Lab 16
Task 2 ----------------------------------

Task 1 **********************************

Task 2 ----------------------------------

Message printed from tick hook interrupt

Task 1 **********************************

Task 1 **********************************

Message printed from tick hook interrupt
```