Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 10/25/2021

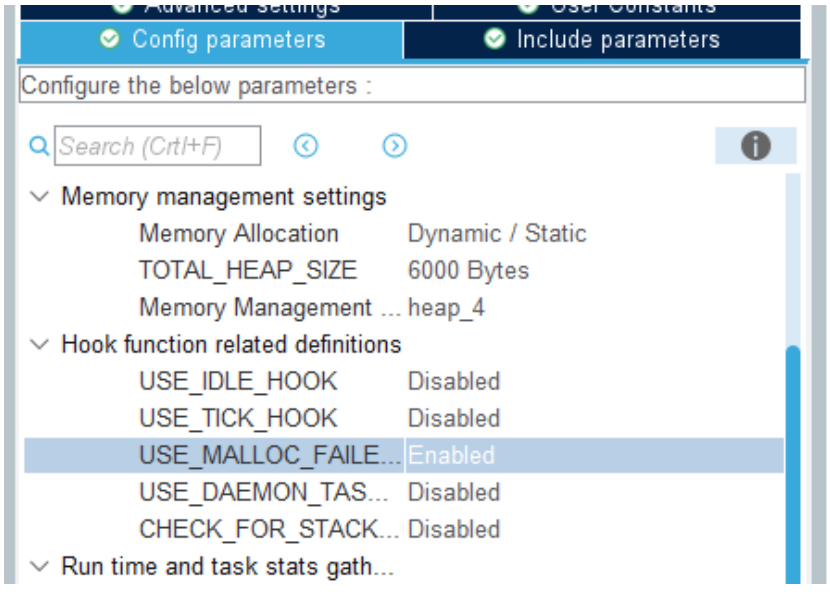
Assignment 2: Memory Management

The following will document completion of the second assignment for ECE-40290, with the stated goals of:

- Create a new STM32CubeIDE Project. (Do NOT use the same project from Lesson 1). Begin by configuring the project the same as in Lesson 1 (e.g. use all the defaults) but then make the changes required for memory allocation as described below. Configure FreeRTOS for a HEAP memory size of 6000. Keep the default dynamic/static allocation along with the default memory 4 allocation option.
- In your default task, blink the LED2 every second and also use pvPortMalloc() to allocate 500 bytes every time you blink the LED2. The goal here is to "leak memory" as your loop runs until you run out of memory.
- Add the vApplicationMallocFailedHook() to your code so that when you run out of memory the code in the vApplication MallocFailedHook() should be a "while (1) { }" that blinks the Wifi/BlE LEDs at a 0.5 second (1/2 second) rate.

1. Create and configure project

To start, we'll open the STM32Cube IDE and create a new project using the same steps as in the previous lesson: selectign the appropriate defaults and ensuring that FREERTOS is selected under the Middleware menu on the configuration interface. From there, we will ensure that TOTAL_HEAP_SIZE is set to 6000 bytes, heap_4 is selected under the memory management option, and that the USE_MALLOC_FAILED_CALLBACK option is set to enabled, all as seen below:



Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 10/25/2021

2. Blink LED2 and allocate memory

From here, we can define the actions to be taken under the function call for *StartDefaultTask()*, seen below. This includes the calls to toggle the green LED, a delay of one second, and the *pvPortMalloc(500)* call to allocate 500 bytes per loop iteration. Additionally, notice the declaration of *size_t memoryAvail*, this was used during debugging to wath the free heap space decrease with each malloc call, though I neglected to capture this with a screenshot.

```
688 /* USER CODE END Header_StartDefaultTask */
689 void StartDefaultTask(void const * argument)
690 {
691   /* USER CODE BEGIN 5 */
692   /* Infinite loop */
693   for(;;)
694   {
695
696       // Variable to track heap space left available
697       size_t memoryAvail = xPortGetFreeHeapSize();
698
699       // Toggle green LED and delay for one second
700       HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
701       osDelay(1000);
702
703       // Allocate 500 bytes every time we iterate through the loop
704       pvPortMalloc(500);
705
706   }
707   /* USER CODE END 5 */
708 }
709
```

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 10/25/2021

3. Implement vApplicationMallocFailedHook()

Finally, we will implement the callback functionality in *vApplicationMallocFailedHook()*, seen below. After enough calls to *pvPortMalloc()* are made in the default task, this function call will trigger, toggling out on the WiFi/BLE LEDs that there is something amiss with the application in a while(1) loop. At this point, the green LED will also cease to toggle.

```
78 /* Private user code ---------------------------------------------------------*/
79 /* USER CODE BEGIN 0 */
80
81 // Overwrite __weak function def w/ LED toggle & delay
82 void vApplicationMallocFailedHook(void)
83 {
84     while (1)
85     {
86         // Toggle WiFi/BLE LEDs and delay for a half second
87         HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
88         osDelay(500);
89     }
90 }
91
92 /* USER CODE END 0 */
```

Closing Thoughts

This proved to be an overall pretty straightforward assignment but served to clearly and concisely demonstrate key memory management techniques using the FreeRTOS API. I'm writing this report a few weeks after completing this and later assignments and can say that I really enjoy the method used by the course and *Mastering the FreeRTOS Real Time Kernel* to present new concepts in bite-sized chunks paired with clear example code.