

## ASSIGNMENT 4(Programming)

Name: K.Vishwanath

Course Title: [Development of Real-Time System](#)

Date: 23-06-2022

---

### Assignment:

Create a task "matrixtask" containing the functionality given in Assignment 2.

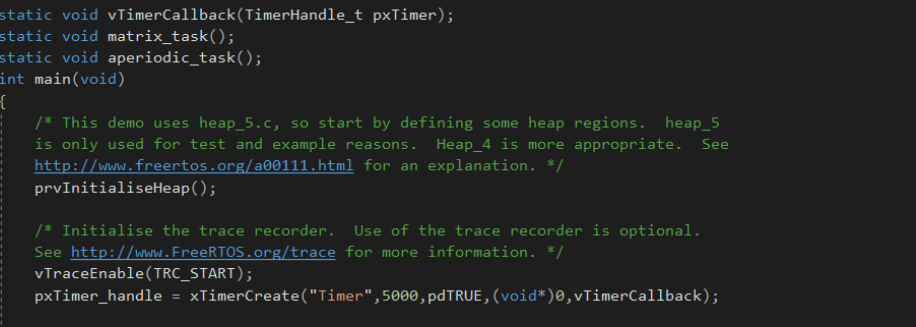
Add a software timer in main() to trigger a software interrupt every 5 seconds.

Define a Timer callback function outside main() with the following functionality:

```
long lExpireCounters = 0;
void vTimerCallback(TimerHandle_t pxTimer)
{
    printf("Timer callback!\n");
    xTaskCreate((pdTASK_CODE)aperiodic_task, (signed char
*) "Aperiodic", configMINIMAL_STACK_SIZE, NULL, 2, &aperiodic_handle);
    long lArrayIndex;
    const long xMaxExpiryCountBeforeStopping = 10;
    configASSERT(pxTimer);
    lExpireCounters += 1;
    /* If the timer has expired 10 times then stop it from running. */
    if (lExpireCounters == xMaxExpiryCountBeforeStopping) {
        /* Do not use a block time if calling a timer API function from a
        timer callback function, as doing so could cause a deadlock! */
        xTimerStop(pxTimer, 0);
    }
}
```

Create an aperiodic task using the following functionality:

```
static void aperiodic_task()
{
    printf("Aperiodic task started!\n");
    fflush(stdout);
    long i;
    for (i = 0; i<1000000000; i++); //Dummy workload
    printf("Aperiodic task done!\n");
    fflush(stdout);
    vTaskDelete(aperiodic_handle);
}
```



```
port.c  main.c  x
Miscellaneous Files - No Configurations  (Global Scope)  vTimerCallback(TimerH

145
146 static void vTimerCallback(TimerHandle_t pxTimer);
147 static void matrix_task();
148 static void aperiodic_task();
149 int main(void)
150 {
151     /* This demo uses heap_5.c, so start by defining some heap regions. heap_5
152     is only used for test and example reasons. Heap_4 is more appropriate. See
153     http://www.freertos.org/a00111.html for an explanation. */
154     prvInitialiseHeap();
155
156     /* Initialise the trace recorder. Use of the trace recorder is optional.
157     See http://www.FreeRTOS.org/trace for more information. */
158     vTraceEnable(TRC_START);
159     pxTimer_handle = xTimerCreate("Timer", 5000, pdTRUE, (void*)0, vTimerCallback);
160
161     if (pxTimer_handle == NULL)
162     {
163         /* The timer was not created. */
164         printf("The timer was not created!\n");
165         fflush(stdout);
166     }
167     else
168     {
169         /* Start the timer. No block time is specified, and
170         even if one was it would be ignored because the RTOS
171         scheduler has not yet been started. */
172         if (xTimerStart(pxTimer_handle, 0) != pdPASS)
173         {
174             /* The timer could not be set into the Active state. */
175             printf("The timer could not be set into the Active state!\n");
176             fflush(stdout);
177         }
178     }
179
180     xTaskCreate(matrix_task, "Matrix", 1024, NULL, 3, &matrix_handle);
181 }
```

The screenshot displays the Visual Studio IDE interface for the RTOS Demo project. The main window shows the source code for the `vTimerCallback` function, which prints the matrix period and task execution time. The right-hand side of the IDE features the Diagnostic Tools window, which provides a summary of the diagnostics session, including events, process memory usage, and CPU usage. The bottom status bar indicates the thread 0x654 has exited with code 0 (0x0).

**Source Code (vTimerCallback):**

```

232     }
233 }
234
235 #if !VFreeRTOSv1001VFreeRTOSDemo(WIN32-MSVC,Debug)RTOSDemo.exe
236 ulMatrixPeriod = 1075;
237 vTaskMatrixPeriod = 1089;
238 }
239 MatrixPeriod = 968;
240 }
241 MatrixPeriod = 961;
242
243 Static void Timer callback!
244 {
245     MatrixPeriod = 1021;
246     printf("Aperiodic task started!
247     fflush(stdout);
248     MatrixPeriod = 1025;
249     long i;
250     MatrixPeriod = 988;
251     for (i = 0; i < 10; i++)
252     {
253         MatrixPeriod = 972;
254         printf("MatrixPeriod = 965;
255         fflush(stdout);
256         Timer callback!
257         MatrixPeriod = 969;
258         fflush(stdout);
259         Aperiodic task started!
260         MatrixPeriod = 965;
261         vTaskMatrixPeriod = 965;
262         MatrixPeriod = 962;
263         }
264     }
265     MatrixPeriod = 950;
266     void vTimerMatrixPeriod = 970;
267     {
268         Timer callback!
269         printf("MatrixPeriod = 998;
270         fflush(stdout);
271         Aperiodic task started!
272         ulAperiodicTaskPeriod = 963;
273         }
274     }
275     No issues found!
276     Aperiodic task done!
277     Response Time is: 2866;
278     Search (Ctrl+E)
279     MatrixPeriod = 964;
280     MatrixPeriod = 957;
281     Name
282     Timer callback!

```

**Diagnostic Tools Summary:**

- Diagnostics session: 39 seconds
- Events: 0
- Process Memory (MB): 2
- CPU (% of all processors): 100

**Summary:**

- Events: 0
- Memory Usage: 0
- CPU Usage: 100

**Thread Information:**

- Thread 0x654: KernelBase.dll, VM64Vntmm.dll, VM64Vsvcrtd.dll
- Thread 0x654: KernelBase.dll, VM64Vntmm.dll, VM64Vsvcrtd.dll
- Thread 0x654: KernelBase.dll, VM64Vntmm.dll, VM64Vsvcrtd.dll

**Thread Exit Information:**

- The thread 0x654 has exited with code 0 (0x0).
- The thread 0x654 has exited with code 0 (0x0).
- The thread 0x654 has exited with code 0 (0x0).

**1. Is the system fast enough to handle all aperiodic tasks? Why?**

Yes, the system is fast enough to handle the aperiodic task.

Because, aperiodic task is generated only when there is a timer interrupt every 5 seconds. But the matrix multiplication only takes 600 to 700ms.

**2. If not, solve this problem without alter the functionality of any task**

I do not see a problem on my machine. In case, if I cannot handle all aperiodic task within next period (5 seconds in this case), I could have created a set priority task and increased the priority of the aperiodic task.

**3. What is the response time of the aperiodic task?**

Response time of the aperiodic task depends on the execution state of Matrix Multiplication task at the time of 5 seconds timer interrupt. For me, I was able to see response time of 400ms on an average.

**4. Provide a screenshot of the running system**

Please find the attached output.png image.