Real Time Operating System
Task Priorities & Deletion

Sherif Hammad

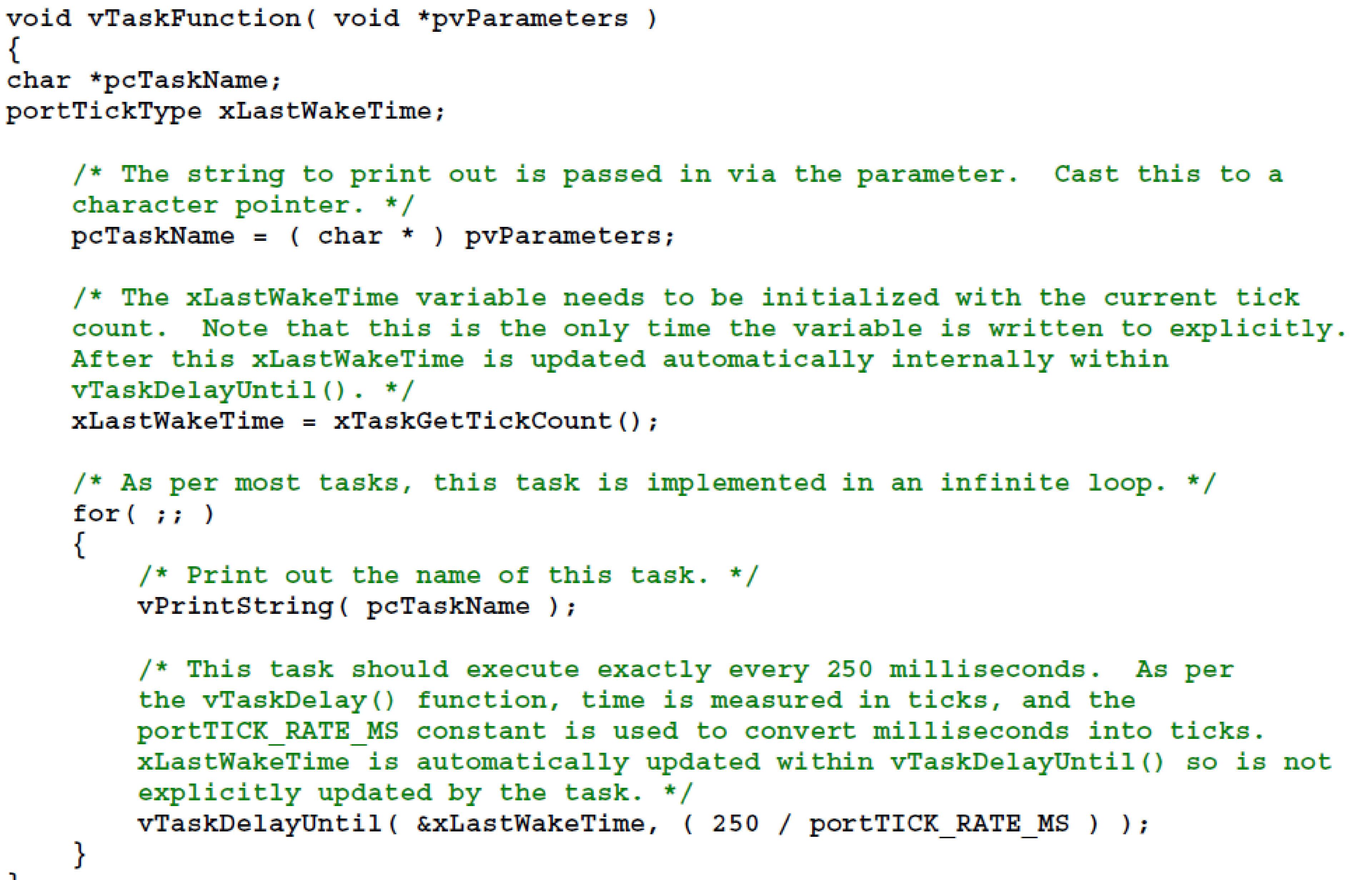
Agenda

- Accurate task periods
- Continuous & Periodic Tasks
- Changing Tasks priorities
- Deleting tasks

Accurate Tasks Periods

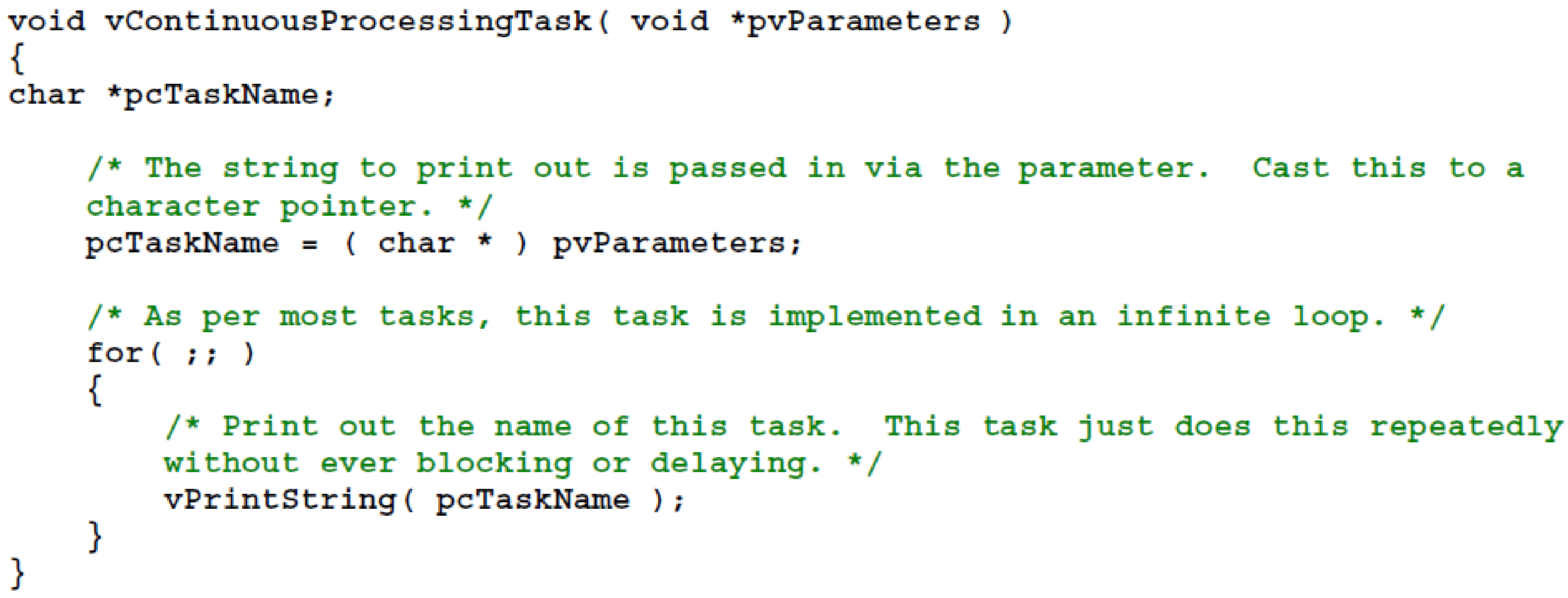
- vTaskDelayUntil() is similar to vTaskDelay() "but"
- vTaskDelay() parameter specifies the number of tick interrupts that should occur between a task calling vTaskDelay() and the same task once again transitioning out of the Blocked state.
- The length of time the task remains in the blocked state is specified by the vTaskDelay() parameter, but the actual time at which the task leaves the blocked state is relative to the time at which vTaskDelay() was called.
- The parameters to vTaskDelayUntil() specify, instead, the exact tick count value at which the calling task should be moved from the Blocked state into the Ready state.
- vTaskDelayUntil() is the API function that should be used when a fixed execution period is required (where you want your task to execute periodically with a fixed frequency), as the time at which the calling task is unblocked is absolute, rather than relative to when the function was called (as is the case with vTaskDelay()).
- vTaskDelayUntil() API function is available only when INCLUDE_vTaskDelayUntil is set to 1 in FreeRTOSConfig.h.

Accurate Tasks Periods (Example 5)

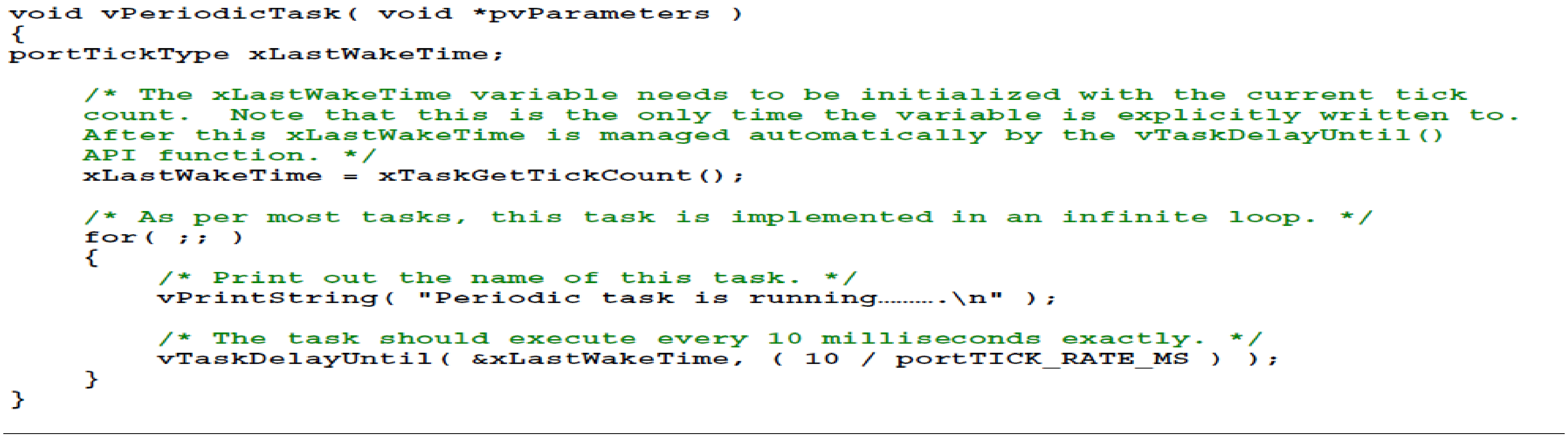


```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
portTickType xLastWakeTime;

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* The xLastWakeTime variable needs to be initialized with the current tick
    count.  Note that this is the only time the variable is written to explicitly.
    After this xLastWakeTime is updated automatically internally within
    vTaskDelayUntil(). */
    xLastWakeTime = xTaskGetTickCount();

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* This task should execute exactly every 250 milliseconds.  As per
        the vTaskDelay() function, time is measured in ticks, and the
        portTICK_RATE_MS constant is used to convert milliseconds into ticks.
        xLastWakeTime is automatically updated within vTaskDelayUntil() so is not
        explicitly updated by the task. */
        vTaskDelayUntil( &xLastWakeTime, ( 250 / portTICK_RATE_MS ) );
    }
}
```

Example 6: Continuous & Periodic Tasks



```
void vContinuousProcessingTask( void *pvParameters )
{
char *pcTaskName;

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task.  This task just does this repeatedly
        without ever blocking or delaying. */
        vPrintString( pcTaskName );
    }
}
```



```
void vPeriodicTask( void *pvParameters )
{
portTickType xLastWakeTime;

    /* The xLastWakeTime variable needs to be initialized with the current tick
    count.  Note that this is the only time the variable is explicitly written to.
    After this xLastWakeTime is managed automatically by the vTaskDelayUntil()
    API function. */
    xLastWakeTime = xTaskGetTickCount();

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( "Periodic task is running……….\n" );

        /* The task should execute every 10 milliseconds exactly. */
        vTaskDelayUntil( &xLastWakeTime, ( 10 / portTICK_RATE_MS ) );
    }
}
```

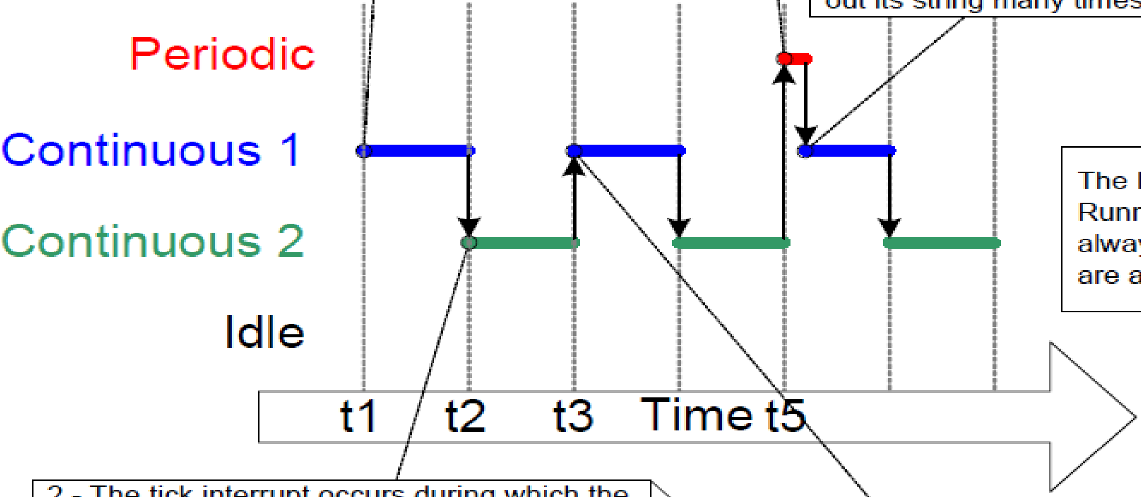
Listing 16. The periodic task used in Example 6

Example 6: Continuous & Periodic Tasks

4 - At time t5 the tick interrupt finds that the Periodic task block period has expired so moved the Periodic task into the Ready state. The Periodic task is the highest priority task so immediately then enters the Running state where it prints out its string exactly once before calling vTaskDelayUntil() to return to the Blocked state.

1 - Continuous task 1 runs for a complete tick period (time slice between times t1 and t2) - during which time it could print out its string many times.

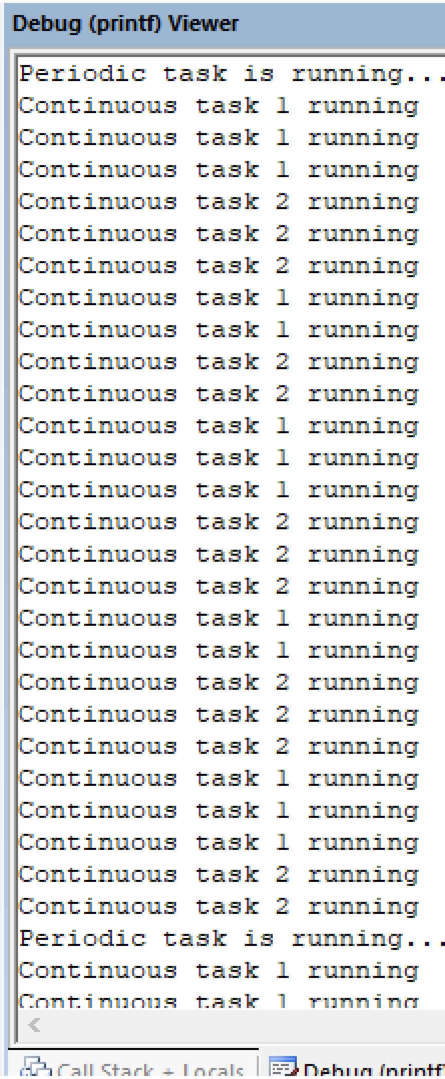
5 - The Periodic task entering the Blocked state means the scheduler has again to choose a task to enter the Running state - in this case Continuous 1 is chosen and it runs up to the next tick interrupt - during which time it could print out its string many times.



The Idle task never enters the Running state as there are always higher priority task that are able to do so.

2 - The tick interrupt occurs during which the scheduler selects a new task to run. As both Continuous tasks have the same priority and both are always able to run the scheduler shares processing time between the two - so Continuous 2 enters the Running state where it remains for the entire tick period - during which time it could print out its string many times.

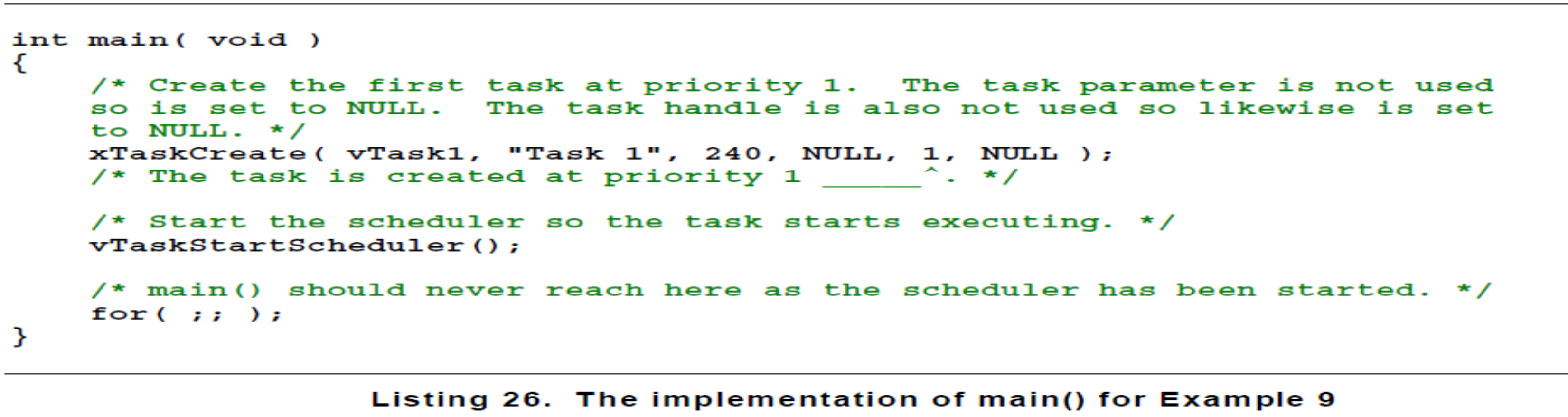
3 - At time t3 the tick interrupt runs again, causing a switch back to Continuous 1, and so it goes on.



```
Periodic task is running...
Continuous task 1 running
Continuous task 1 running
Continuous task 1 running
Continuous task 2 running
Continuous task 2 running
Continuous task 2 running
Continuous task 1 running
Continuous task 1 running
Continuous task 2 running
Continuous task 2 running
Continuous task 1 running
Continuous task 1 running
Continuous task 1 running
Continuous task 2 running
Continuous task 2 running
Continuous task 2 running
Continuous task 1 running
Continuous task 1 running
Continuous task 2 running
Continuous task 2 running
Continuous task 2 running
Continuous task 1 running
Continuous task 1 running
Continuous task 1 running
Continuous task 2 running
Continuous task 2 running
Periodic task is running...
Continuous task 1 running
Continuous task 1 running
```

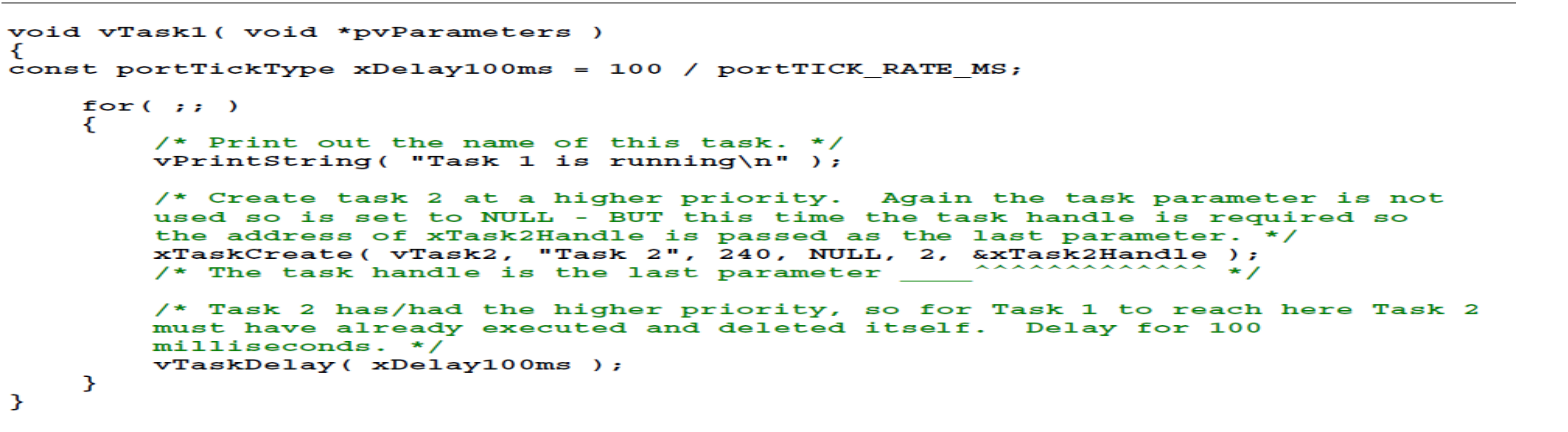
Figure 12. The execution pattern of Example 6

Deleting a Task; Example 9

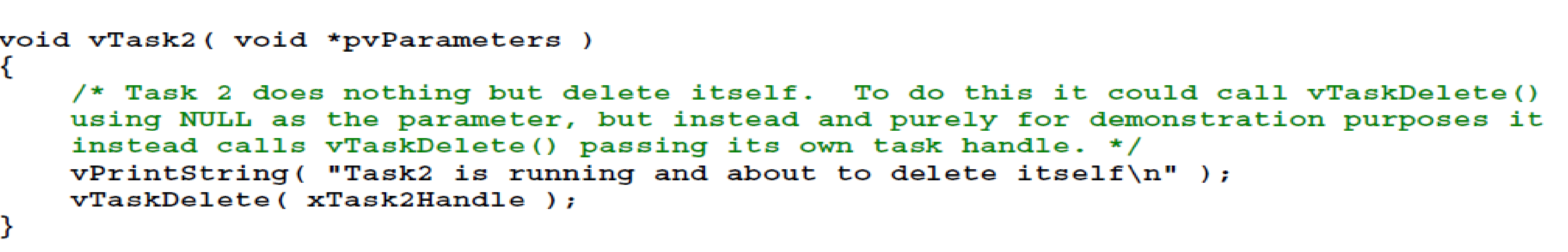


```
int main( void )
{
    /* Create the first task at priority 1.  The task parameter is not used
    so is set to NULL.  The task handle is also not used so likewise is set
    to NULL. */
    xTaskCreate( vTask1, "Task 1", 240, NULL, 1, NULL );
    /* The task is created at priority 1 _____^. */

    /* Start the scheduler so the task starts executing. */
    vTaskStartScheduler();

    /* main() should never reach here as the scheduler has been started. */
    for( ;; );
}
```

Listing 26. The implementation of main() for Example 9



```
void vTask1( void *pvParameters )
{
const portTickType xDelay100ms = 100 / portTICK_RATE_MS;

    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( "Task 1 is running\n" );

        /* Create task 2 at a higher priority.  Again the task parameter is not
        used so is set to NULL - BUT this time the task handle is required so
        the address of xTask2Handle is passed as the last parameter. */
        xTaskCreate( vTask2, "Task 2", 240, NULL, 2, &xTask2Handle );
        /* The task handle is the last parameter _____^^^^^^^^^^^^^^ */

        /* Task 2 has/had the higher priority, so for Task 1 to reach here Task 2
        must have already executed and deleted itself.  Delay for 100
        milliseconds. */
        vTaskDelay( xDelay100ms );
    }
}
```

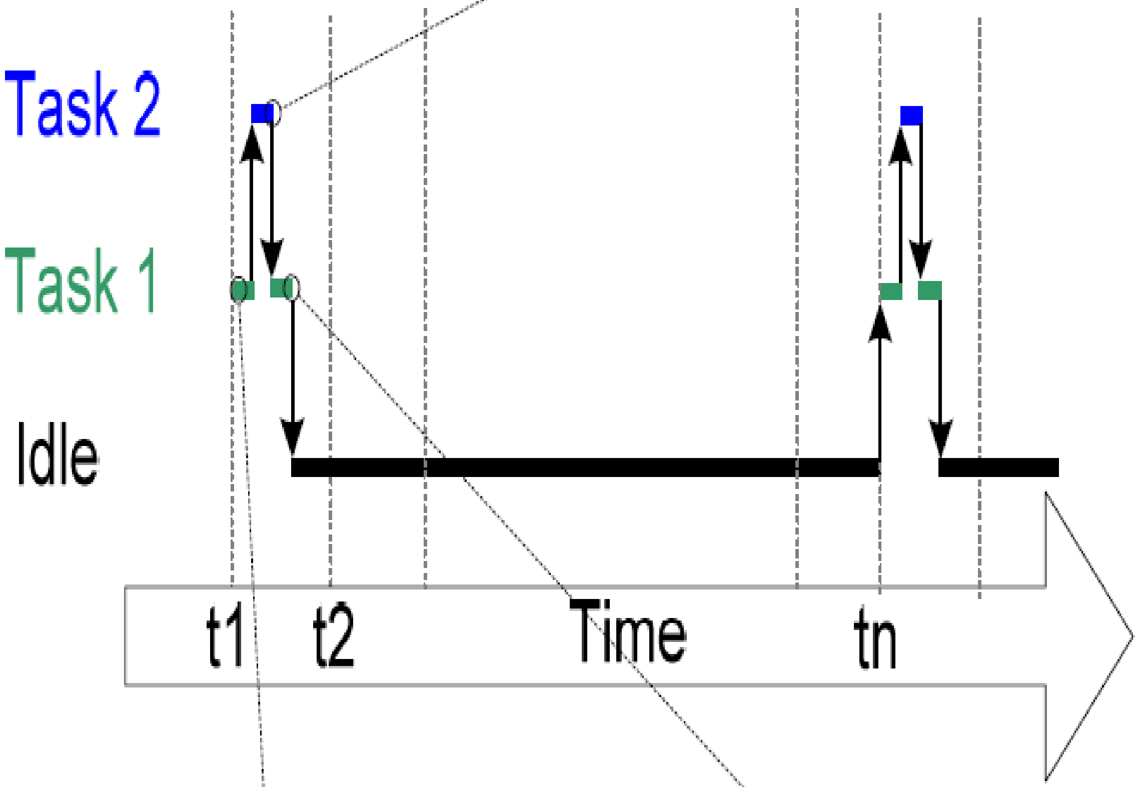


```
void vTask2( void *pvParameters )
{
    /* Task 2 does nothing but delete itself.  To do this it could call vTaskDelete()
    using NULL as the parameter, but instead and purely for demonstration purposes it
    instead calls vTaskDelete() passing its own task handle. */
    vPrintString( "Task2 is running and about to delete itself\n" );
    vTaskDelete( xTask2Handle );
}
```

Deleting a Task; Example 9

Task 2 Preempts 1

2 - Task 2 does nothing other than delete itself, allowing execution to return to Task 1.

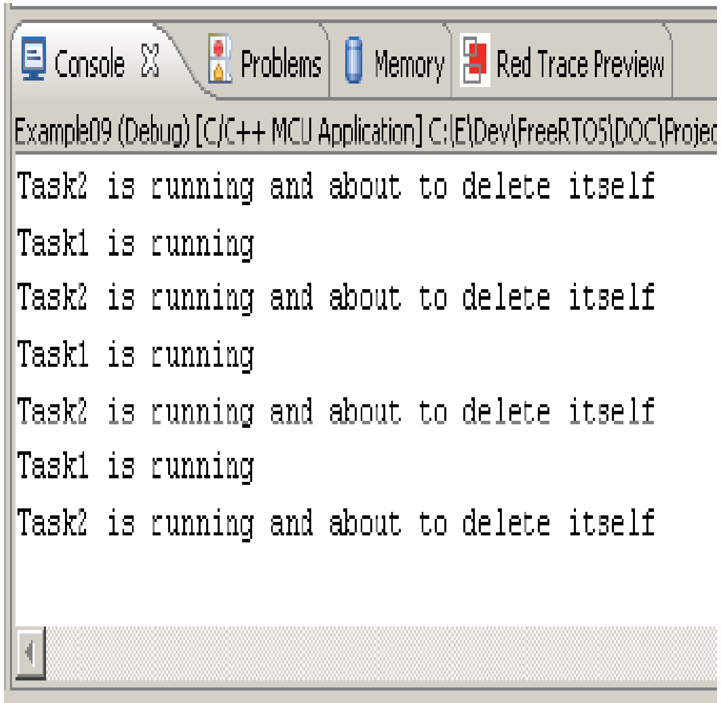


1 - Task 1 runs and creates Task 2. Task 2 starts to run immediately as it has the higher priority.

3 - Task 1 calls vTaskDelay(), allowing the idle task to run until the delay time expires, and the whole sequence repeats.

Task 1 Resumes

Figure 17. The execution sequence for Example 9



```
Example09 (Debug) [C/C++ MCU Application] C:\E\Dev\FreeRTOS\DOC\Projec
Task2 is running and about to delete itself
Task1 is running
Task2 is running and about to delete itself
Task1 is running
Task2 is running and about to delete itself
Task1 is running
Task2 is running and about to delete itself
```

Priorities & Preemption

- 1_three_tasks_same_priority
- 2_three_tasks_different_priority
- 3_three_tasks_preemption_delete_task (Example 9)