FreeRTOS Scheduler Konfiguration

Table 14. The FreeRTOSConfig.h settings that configure the kernel to use Prioritized Pre-emptive Scheduling with Time Slicing

Constant	Value
configUSE_PREEMPTION	1
configUSE_TIME_SLICING	1

HM

Zwei “identische” freeRTOS-Tasks

```
void vTask2( void *pvParameters )
{
const char *pcTaskName = "Task 2 is running\r\n";
volatile uint32_t ul; /* volatile to ensure ul is not optimized away. */

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

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later examples will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

Listing 15. Implementation of the second task used in Example 1

```
        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later examples will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

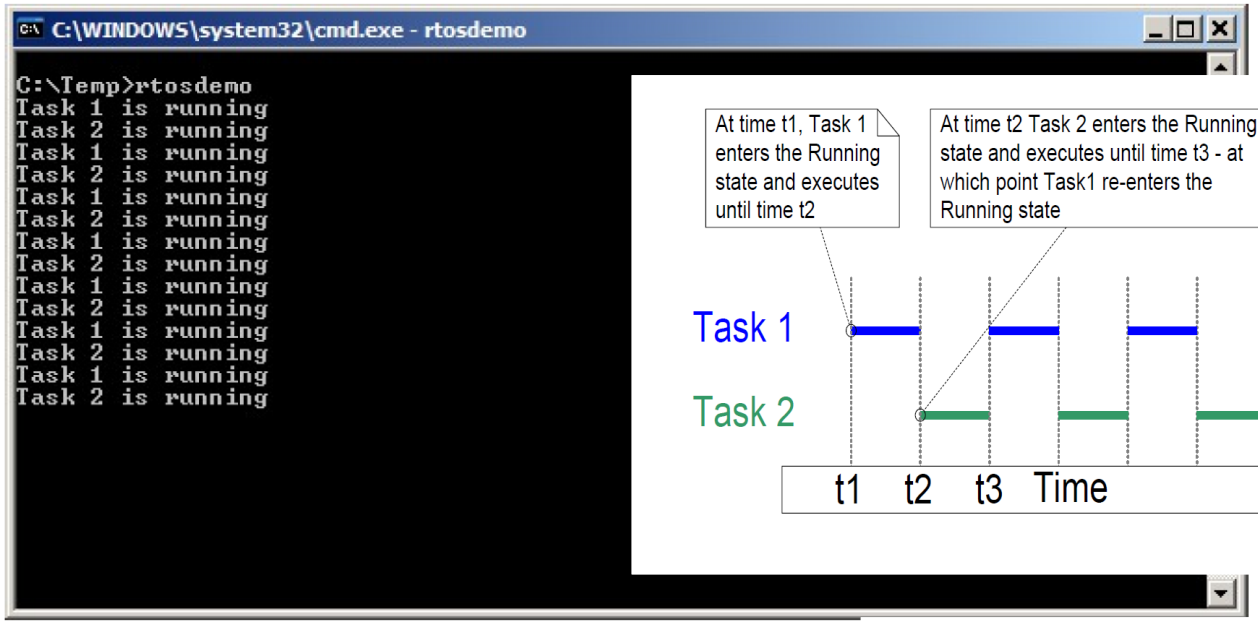
Listing 14. Implementation of the first task used in Example 1

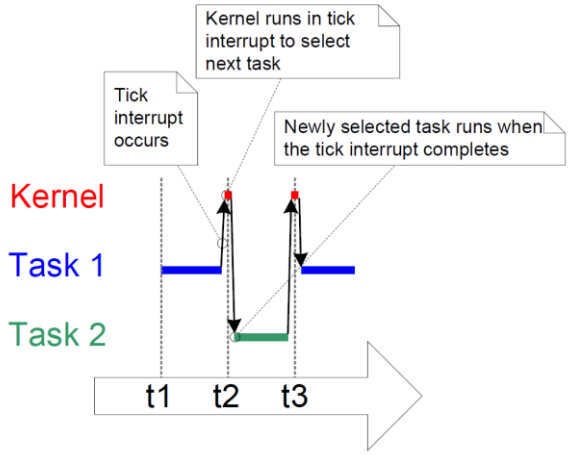
Zwei Tasks erzeugen mit gleicher Priorität

```
int main( void )
{
    /* Create one of the two tasks.  Note that a real application should check
    the return value of the xTaskCreate() call to ensure the task was created
    successfully. */
    xTaskCreate(    vTask1,  /* Pointer to the function that implements the task. */
                    "Task 1",/* Text name for the task.  This is to facilitate
                             debugging only. */
                    1000,    /* Stack depth - small microcontrollers will use much
                             less stack than this. */
                    NULL,    /* This example does not use the task parameter. */
                    1,       /* This task will run at priority 1. */
                    NULL );  /* This example does not use the task handle. */

    /* Create the other task in exactly the same way and at the same priority. */
    xTaskCreate( vTask2, "Task 2", 1000, NULL, 1, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* If all is well then main() will never reach here as the scheduler will
    now be running the tasks.  If main() does reach here then it is likely that
    there was insufficient heap memory available for the idle task to be created.
    Chapter 2 provides more information on heap memory management. */
    for( ;; );
}
```

Listing 16. Starting the Example 1 tasks

Verhalten Tasks mit gleicher Priorität





Tasks mit unterschiedlicher Priorität

```
/* Define the strings that will be passed in as the task parameters.  These are
defined const and not on the stack to ensure they remain valid when the tasks are
executing. */
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\r\n";

int main( void )
{
    /* Create the first task at priority 1.  The priority is the second to last
    parameter. */
    xTaskCreate( vTaskFunction, "Task 1", 1000, (void*)pcTextForTask1, 1, NULL );

    /* Create the second task at priority 2, which is higher than a priority of 1.
    The priority is the second to last parameter. */
    xTaskCreate( vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 2, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* Will not reach here. */
    return 0;
}
```

Listing 21. Creating two tasks at different priorities

Verhalten Tasks mit unterschiedlicher Priorität (beide Tasks immer “Ready”)

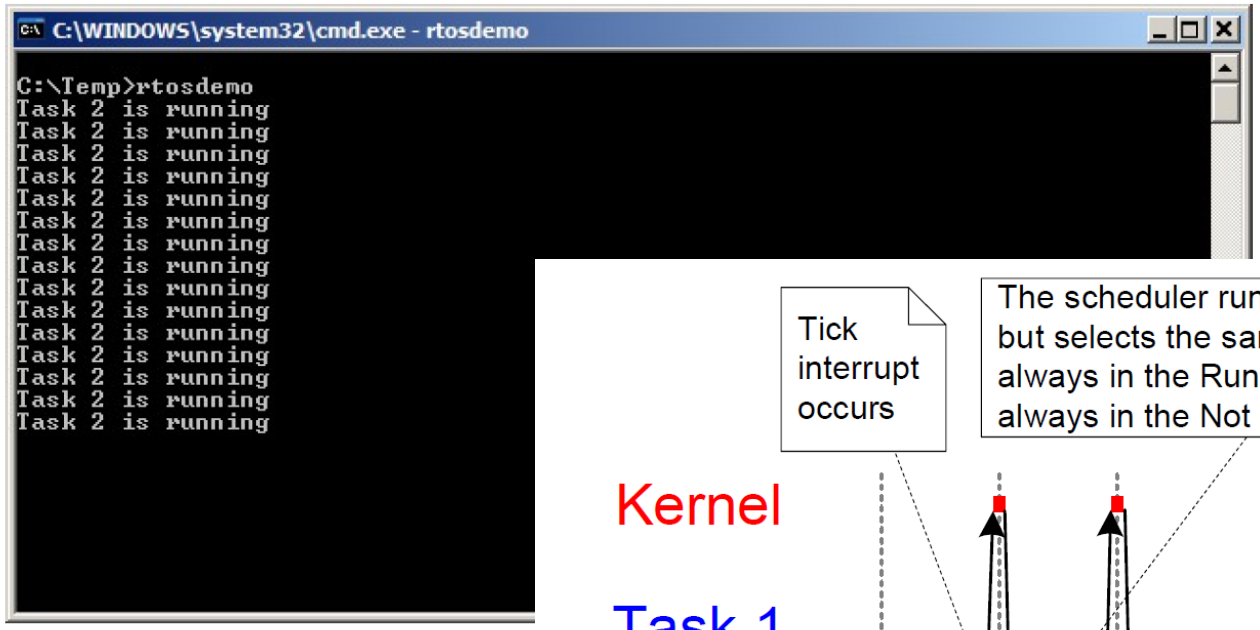
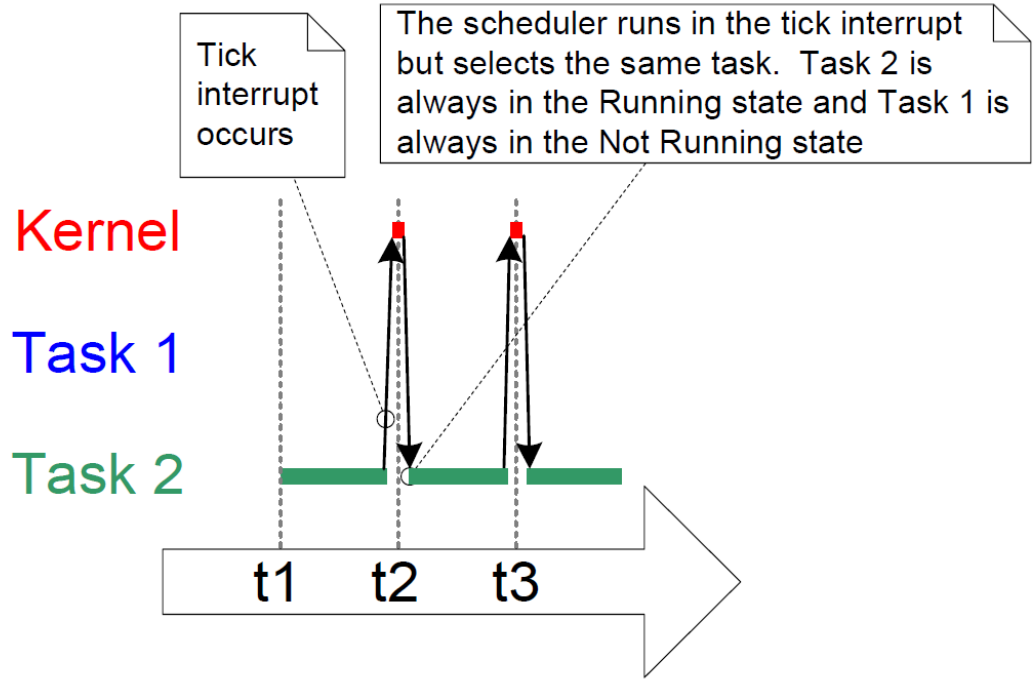


Figure 13. Running I



FreeRTOS Task States

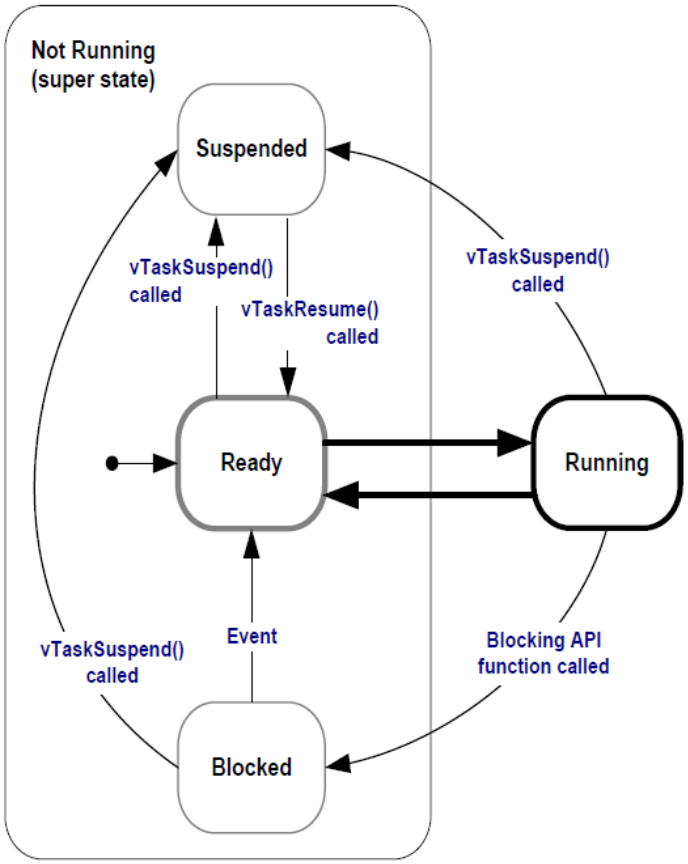


Figure 15. Full task state machine

Task in “blocked”-Zustand versetzen

```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
const TickType_t xDelay250ms = pdMS_TO_TICKS( 250 );

    /* 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. */
        vPrintString( pcTaskName );

        /* Delay for a period.  This time a call to vTaskDelay() is used which places
        the task into the Blocked state until the delay period has expired.  The
        parameter takes a time specified in 'ticks', and the pdMS_TO_TICKS() macro
        is used (where the xDelay250ms constant is declared) to convert 250
        milliseconds into an equivalent time in ticks. */
        vTaskDelay( xDelay250ms );
    }
}
```

Listing 23. The source code for the example task after the null loop delay has been replaced by a call to vTaskDelay()

HM

Verhalten wenn Tasks “blocked”

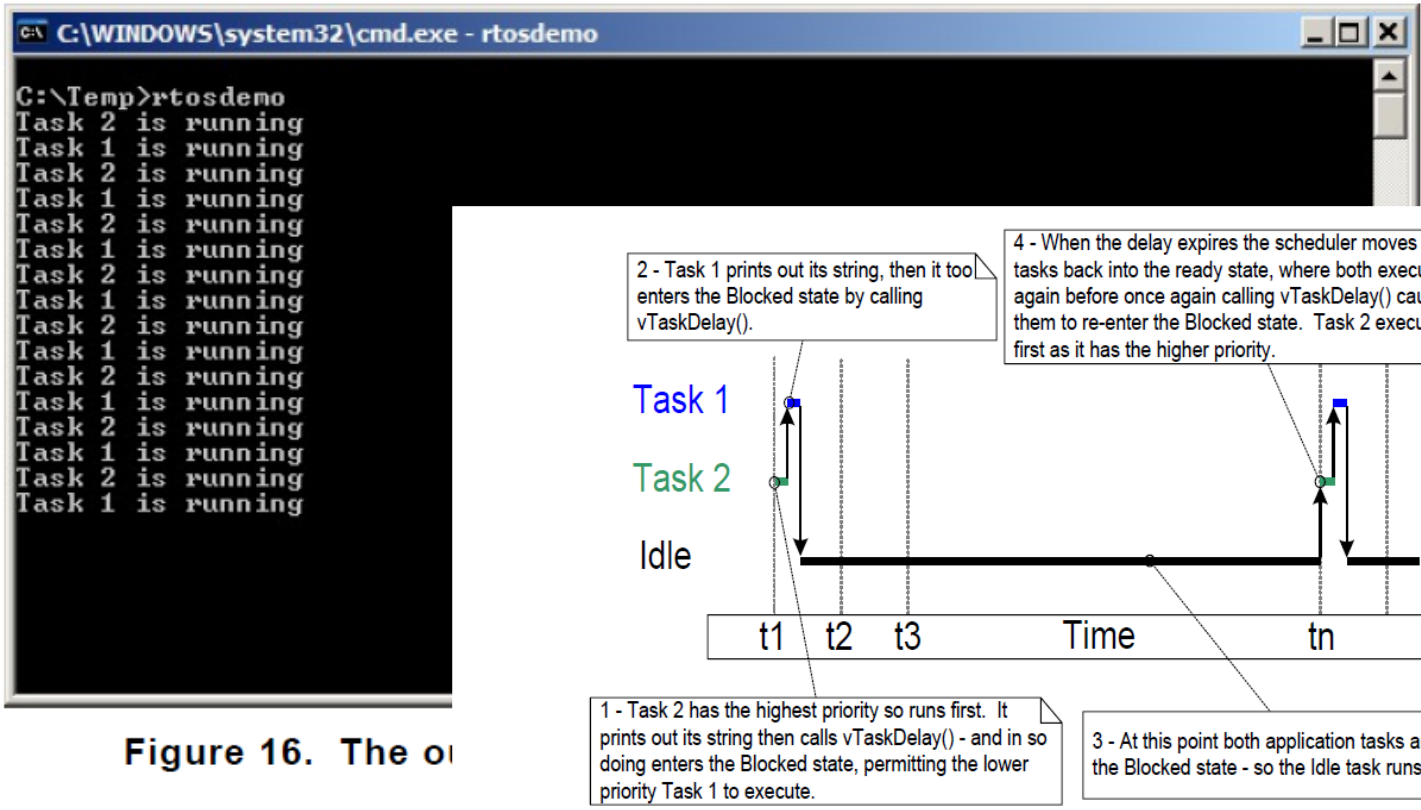


Figure 16. The o

Figure 17. The execution sequence when the tasks use vTaskDelay() in place of the NULL loop