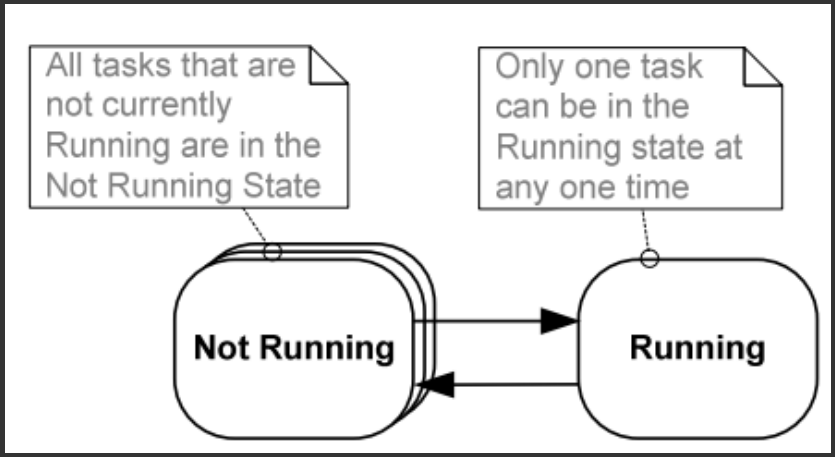
>>> Operating Systems And Applications For Embedded Systems
>>> FreeRTOS

Name: Mariusz Naumowicz
Date: 27 sierpnia 2018

>>> Plan

1. FreeRTOS
   - TOP LEVEL TASK STATES
   - Creating Tasks
   - The actual execution pattern of the two tasks
   - Tick interrupt executing
   - The execution pattern when one task has a higher priority than the other
   - Full task state machine
   - The execution sequence when the tasks use vTaskDelay() in place of the NULL loop
   - The execution pattern with periodic task
   - The sequence of task execution without idle state
   - The execution sequence with task deleting
   - Execution pattern with pre-emption points highlighted
2. Interrupt Management
   - Interrupt example
3. Memory Management
   - RAM allocation

>>> TOP LEVEL TASK STATES



Listing 1: Listing

```
void vTask1( void *pvParameters )
{
const char *pcTaskName = "Task 1 is running\r\n";
volatile unsigned long ul;
/* 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. */
}
}
```

>>> Creating Tasks II

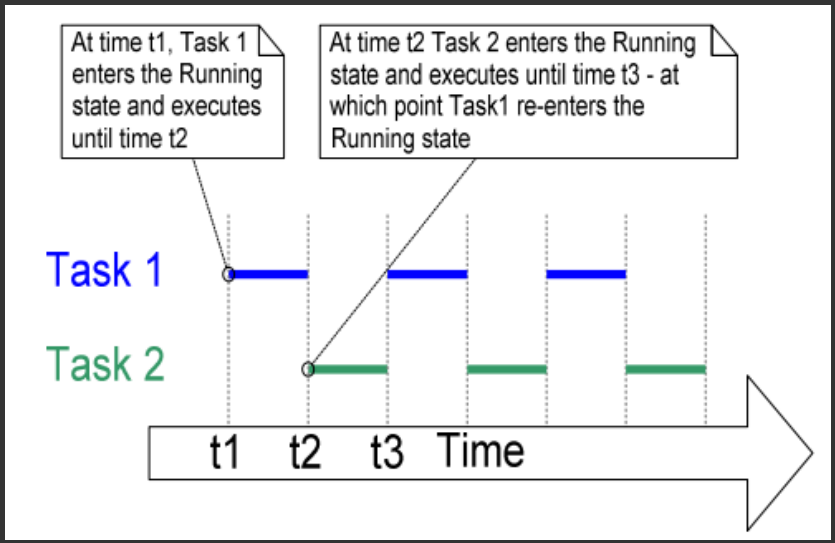
```
}
void vTask2( void *pvParameters )
{
const char *pcTaskName = "Task 2 is running\r\n";
volatile unsigned long ul;
/* As per most tasks, this task is implemented in an infinite loop. */
for( ;; )
{
/* Print out the name of this task. */
vPrintString( pcTaskName );
/* Delay for a period. */
for( ul = 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. */
}
}
}
```

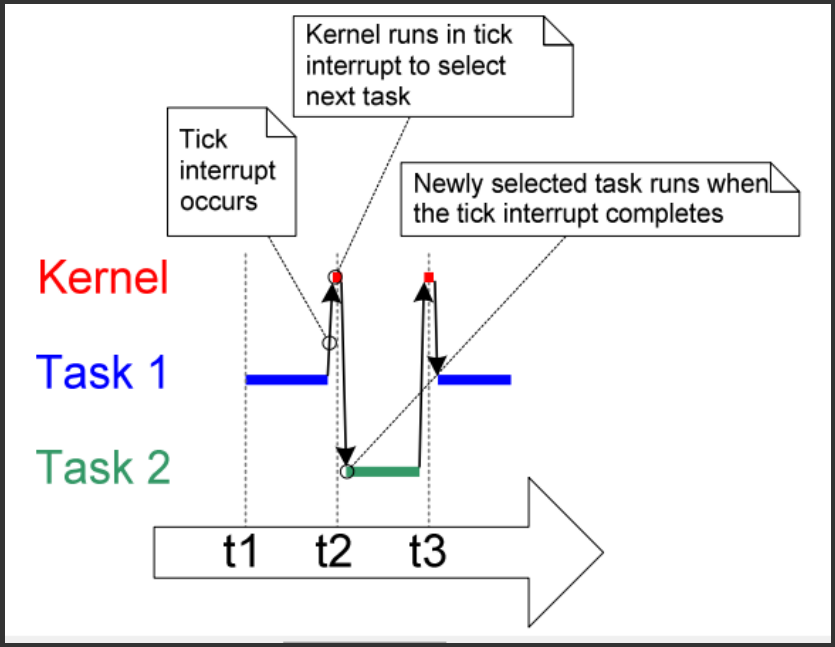
>>> Creating Tasks III

```
int main( void )
{
/* Create one of the two tasks. Note that a real application should che
the return value of the xTaskCreate() call to ensure the task was creat
successfully. */
xTaskCreate( vTask1, /* Pointer to the function that implements the tas
"Task 1",/* Text name for the task. This is to facilitate debugging
only. */
1000, /* Stack depth - most small microcontrollers will use much
less stack than this. */
NULL, /* We are not using the task parameter. */
1, /* This task will run at priority 1. */
NULL ); /* We are not going to use the task handle. */
/* Create the other task in exactly the same way and at the same priori
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 wi
now be running the tasks. If main() does reach here then it is likely t
```

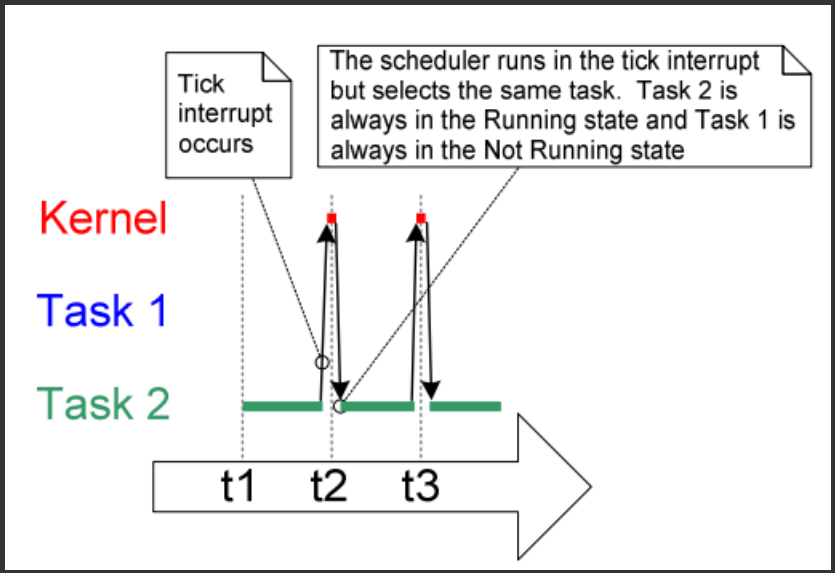
```
there was insufficient heap memory available for the idle task to be cr
for( ;; );
}
```

>>> The actual execution pattern of the two tasks

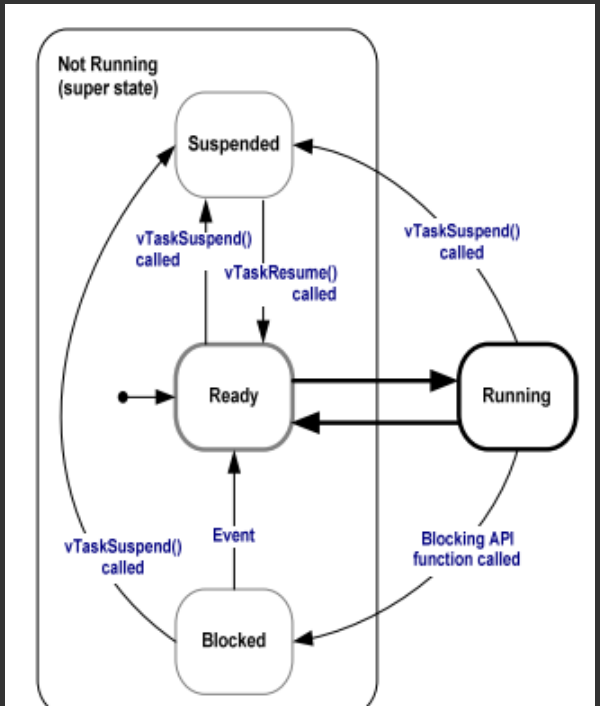



Kernel runs in tick interrupt to select next task
Tick interrupt occurs
Newly selected task runs when the tick interrupt completes
Kernel
Task 1
Task 2
t1
t2
t3

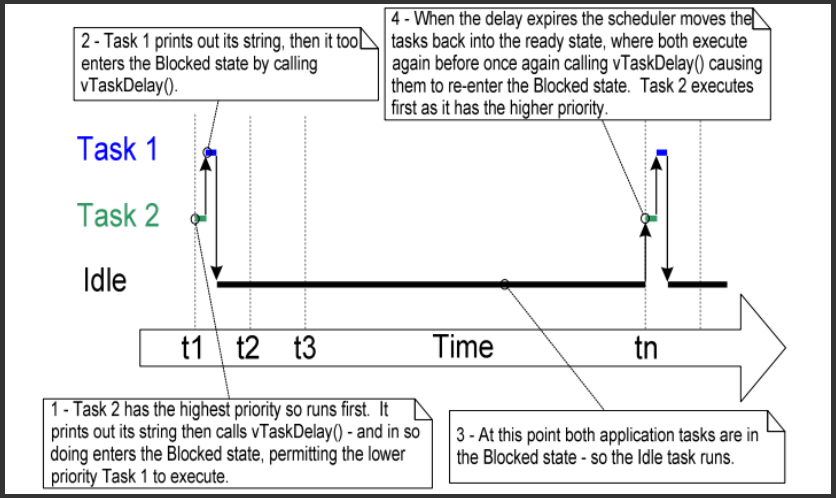
>>> The execution pattern when one task has a higher priority than the other



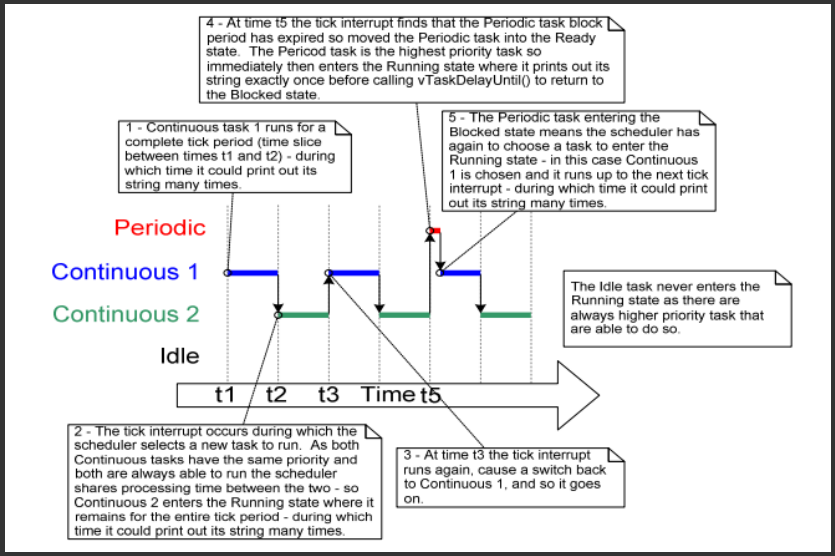
>>> Full task state machine



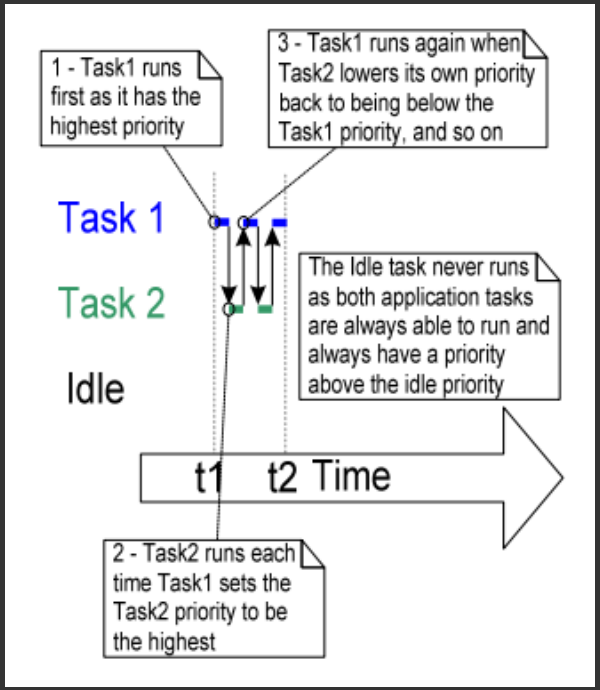
>>> The execution sequence when the tasks use vTaskDelay() in place of the NULL loop



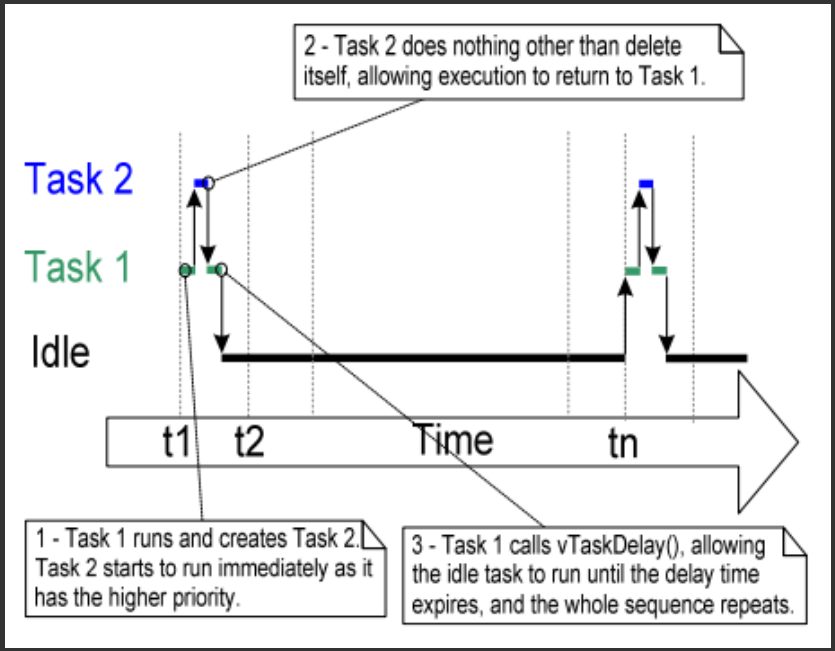
>>> The execution pattern with periodic task



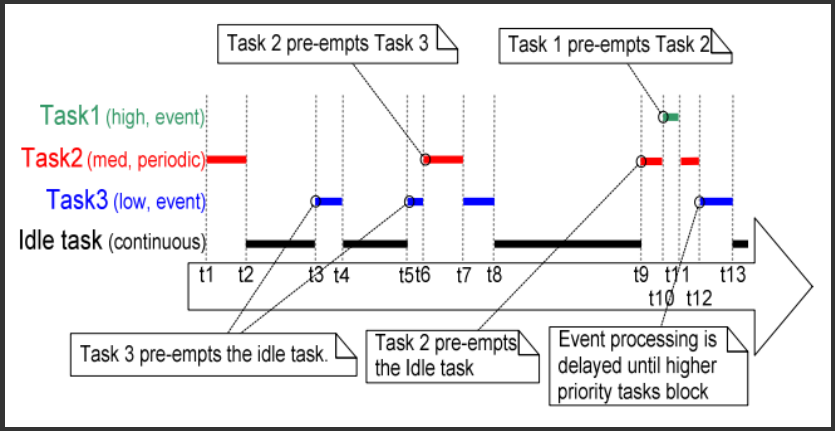
>>> The sequence of task execution without idle state



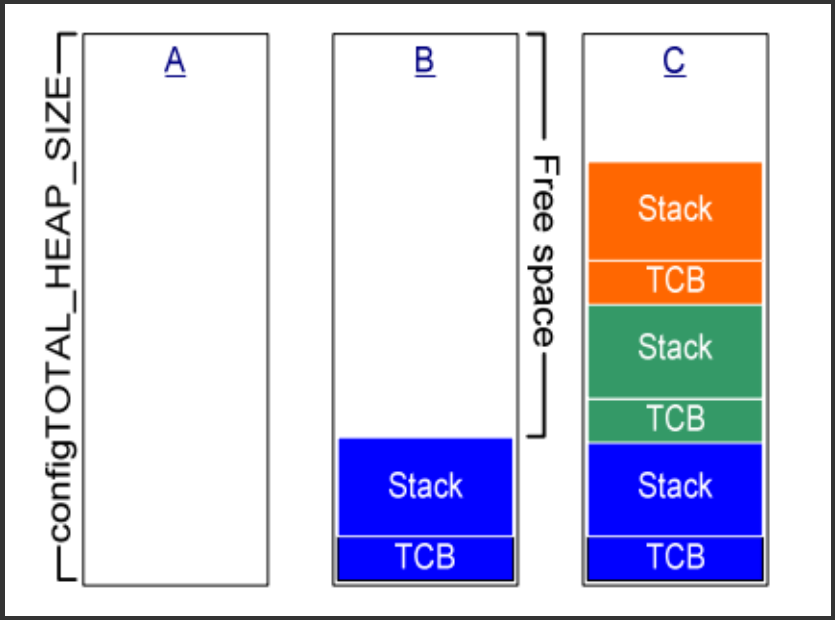
>>> The execution sequence with task deleting

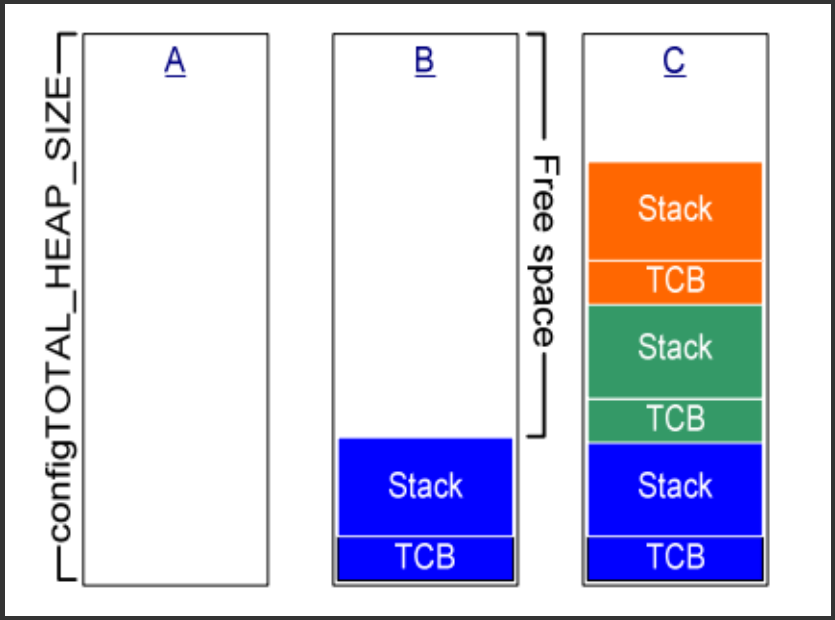


>>> Execution pattern with pre-emption points highlighted



>>> Interrupt example




A
B
C
configTOTAL_HEAP_SIZE
Free space
Stack
TCB
Stack
TCB
Stack
TCB
Stack
TCB

>>> References

R. Barry.
Using the FreeRTOS Real Time Kernel: A Practical Guide.
Real Time Engineers Limited, 2010.