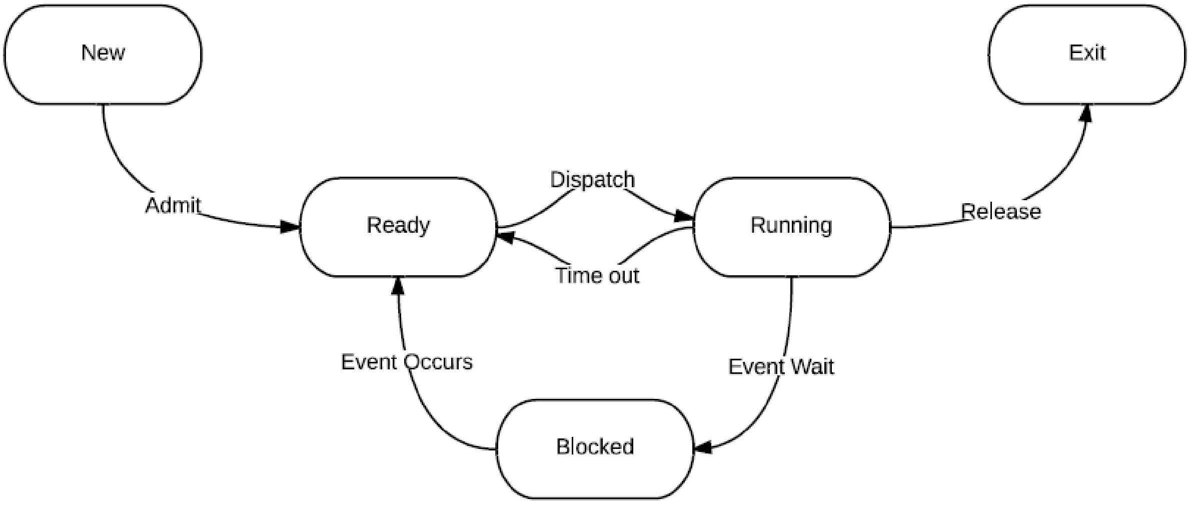
CG2271 Real-Time Operating Systems

Tutorial 4

In this tutorial, we are going to cover many of the important aspects of MultiTasking through the Lab 6 Manual. Almost all the answers for the Lab Manual (except for the last part) are covered here. The objective is for you to have a very clear understanding of how a multi-threaded program works and to be able to analyse it.

Q1. Let's first look back at the State Transition Diagram.



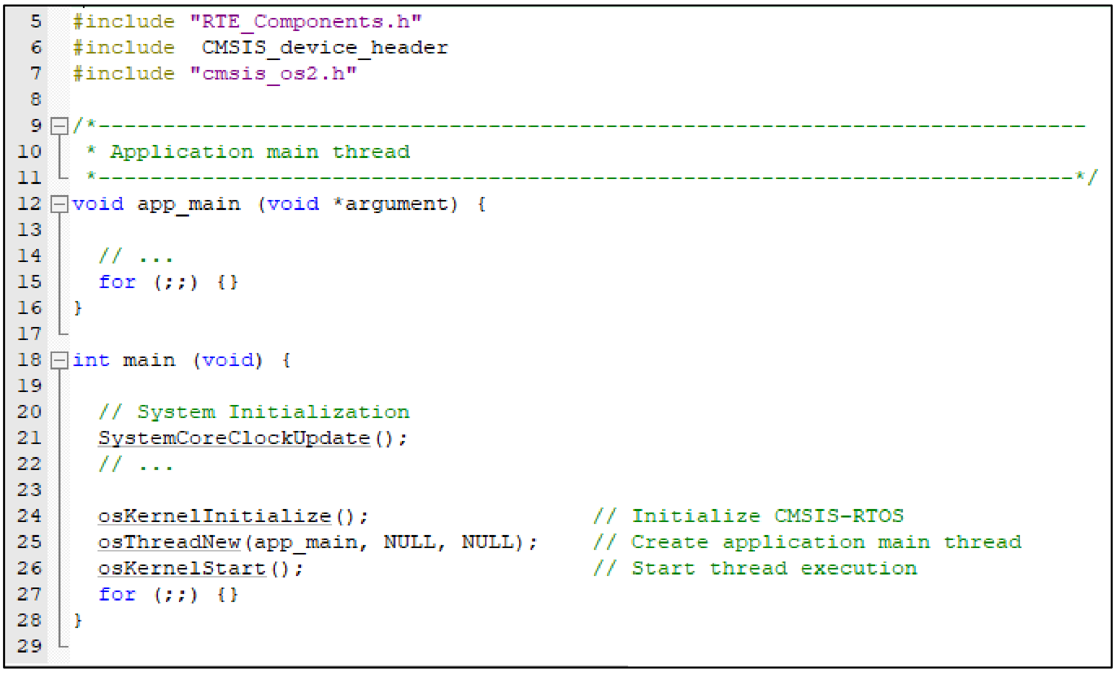
a. Task A is Running and chooses to give up the CPU voluntarily, what state does it go to?

b. Task A is Running and a higher priority Task B becomes Ready. What will happen?

c. After some time, Task B requests for some resource and is unable to acquire it. It is unable to proceed without this resource. What happens?

d. After 5ms, the resource required by Task B is available. What happens?

Q2. The following code snippet shows the way in which a task is created in RTX.



```
#include "RTE_Components.h"
#include  CMSIS_device_header
#include "cmsis_os2.h"

/*----------------------------------------------------------------------------
 * Application main thread
 *---------------------------------------------------------------------------*/
void app_main (void *argument) {

  // ...
  for (;;) {}
}

int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  // ...

  osKernelInitialize();                 // Initialize CMSIS-RTOS
  osThreadNew(app_main, NULL, NULL);    // Create application main thread
  osKernelStart();                      // Start thread execution
  for (;;) {}
}
```

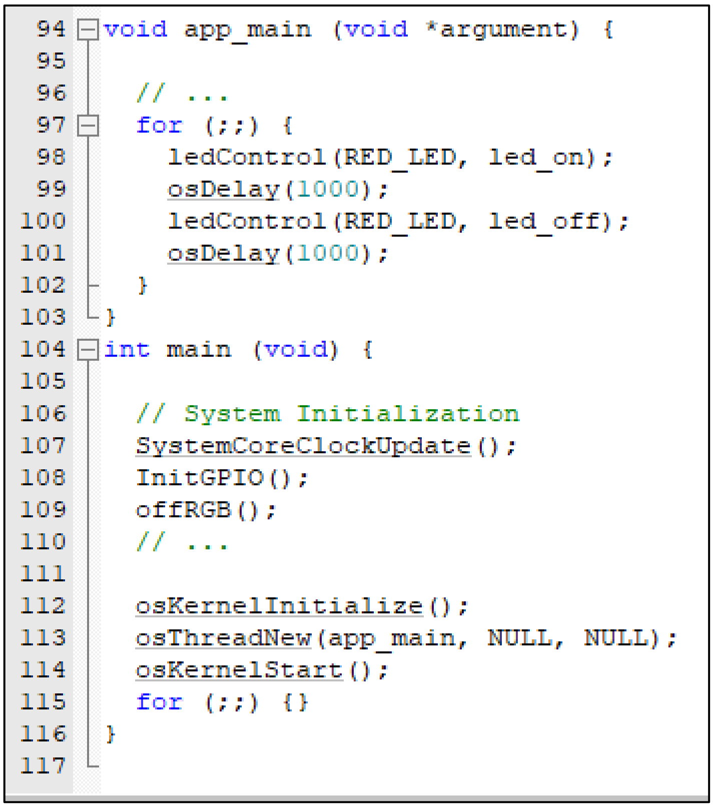
a. The OS call, osThreadNew() takes in three parameters. What are they?

b. When will app_main() be called?

c. Why is there a need for the “for(;;) { }” loop in the app_main().

Q3. Exploring the Blinky Function

Examine the following code snippet.



```
void app_main (void *argument) {

  // ...
  for (;;) {
    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);
  }
}
int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  InitGPIO();
  offRGB();
  // ...

  osKernelInitialize();
  osThreadNew(app_main, NULL, NULL);
  osKernelStart();
  for (;;) {}
}

```

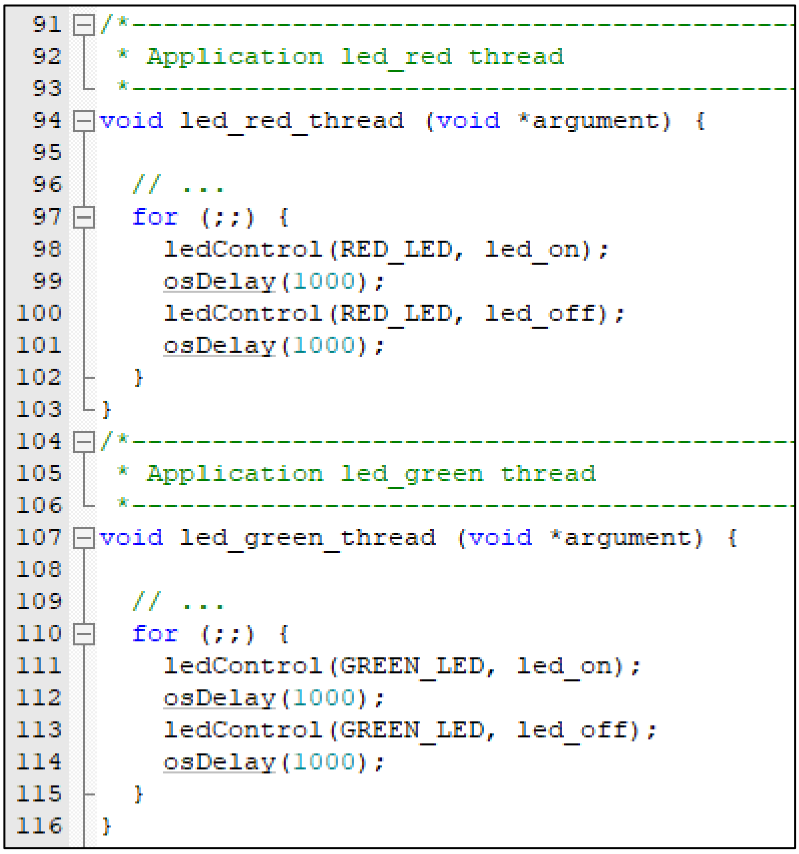
a. When we call osDelay() what happens to the app_main() task?

b. What will the CPU execute during that delay time?

c. If we use a normal delay() routine like what you have been doing so far, will we see the same effect?

Q4. Double Blinky

The following code snippet shows you TWO tasks each controlled a single colour of the led.



```
/*----------------------------------------------------------------------------
 * Application led_red thread
 *---------------------------------------------------------------------------*/
void led_red_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);
  }
}
/*----------------------------------------------------------------------------
 * Application led_green thread
 *---------------------------------------------------------------------------*/
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(GREEN_LED, led_on);
    osDelay(1000);
    ledControl(GREEN_LED, led_off);
    osDelay(1000);
  }
}
```

a. What would be the expected behaviour?

b. Can you draw a timeline to show what happens? Your timeline must clearly show the state of the tasks as they are executing.