CG2271 Real-Time Operating Systems

Tutorial 5 Suggested Solutions

In this tutorial, we are going to cover many of the important aspects of Scheduling and Concurrency.

Q1. Assume three periodic tasks T1 = (3, 9, 9), T2 = (5, 18, 18), and T3 = (4, 12, 12) where the information represents (WCET, Deadline, Period). Demonstrate that a feasible schedule based on fixed priorities exists or prove that one cannot exist.

Answer: If there is a feasible scheduled based on fixed priority, then RMS can generate a feasible schedule. Let us try to schedule these tasks with RMS. T1 has higher priority than T3 and T3 has higher priority than T2.

Cycle	Task	
1.	T1	
2.	T1	
3.	T1	
4.	T3	
5.	T3	
6.	T3	
7.	T3	
8.	T2	
9.	T2	
10.	T1	→ P1 ready
11.	T1	
12.	T1	
13.	T3	→ P3 ready
14.	T3	
15.	T3	
16.	T3	
17.	T2	
18.	T2	

T2 will miss its deadline as we can execute T2 for only 4 cycles by its deadline of 18 cycles.

Q2. Given a set of tasks with the following specifications:

Task	C_i	P_i
T1	2	6
T2	3	8
T3	4	15

a) What is the CPU utilization of this set of tasks?

Answer:

U = 2/6 + 3/8 + 4/15

= 0.975

b) Is this set of tasks RMS schedulable under the utilization bound criteria?

Answer:

Threshold in utilization bound = $3(2^{1/3} - 1)$

= 0.78

Since U > 0.78, task set may or may not be RMS schedulable.

c) Schedule the tasks using RMS

Time	Task	Period
1	T1	
2	T1	
3	T2	
4	T2	
5	T2	
6	T3	
7	T1	T1
8	T1	
9	T2	T2
10	T2	
11	T2	
12	T3	
13	T1	T1
14	T1	
15	T3	

P3 missed deadline.

Q3. In the Lab, you explored the use of Mutexes to control access to the shared resource, the RGB LED.

This is the snippet of the thread with the use of the mutex.

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

  // ...
  for (;;) {
    osMutexAcquire(myMutex, osWaitForever);

    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);

    osMutexRelease(myMutex);
  }
}
```

The second parameter for osMutexAcquire() is a timeout value which is currently set to osWaitForever. What if that value is now changed to 0.

a) What is the significance of this change?

Answer:

If the value of the timeout is now changed to 0, then the thread will not go into a BLOCKED state if the resource became unavailable. In this case, the code must be modified to ensure that there are options to handle the different scenarios.

b) Show how the code in the thread must be modified with the timeout value changed to 0?

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

  osStatus_t myStatus;
  // ...
  for (;;) {
    myStatus = osMutexAcquire(myMutex, 0);

    if(myStatus != osOK)
    {
      // code to handle this case
    }
    else
    {
      ledControl(RED_LED, led_on);
      osDelay(1000);
      ledControl(RED_LED, led_off);
      osDelay(1000);

      osMutexRelease(myMutex);
    }
  }
}
```

THE END