

ECEN 5623

REAL-TIME EMBEDDED

SYSTEMS

HOMEWORK 3

SUBMITTED BY:

SOWMYA

RAMAKRISHNAN

108684769

Using FreeRTOS running on the TI TIVA board, problems 10.1, 10.2 and 10.3 from the text were done.

Porting of FreeRTOS was done using the FreeRTOS-Documentation provided in D2L.

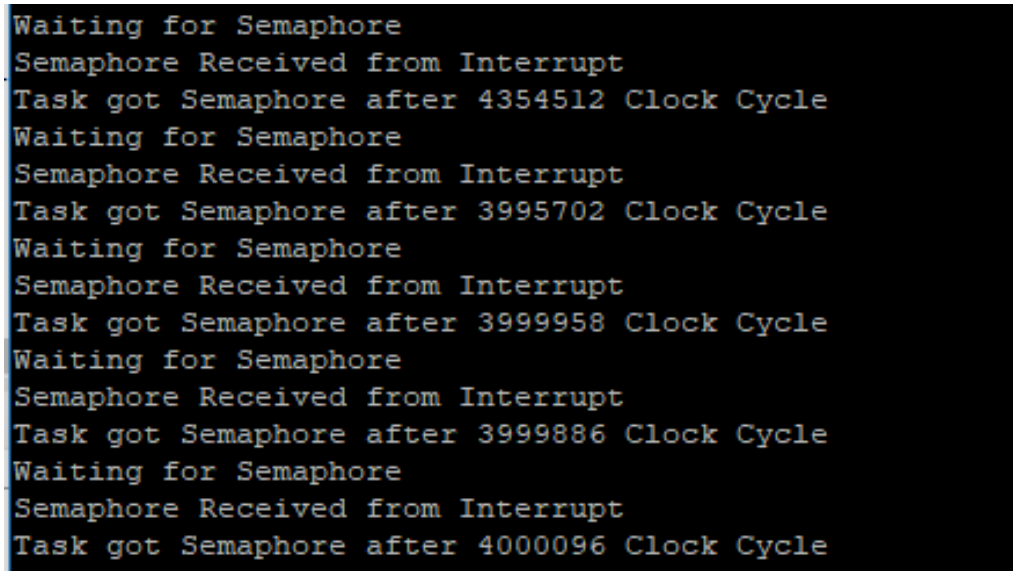
The FreeRTOS Reference Manual and Mastering the FreeRTOS Real Time Kernel were read and the FreeRTOS API were understood.

1.

Create a user-defined interrupt handler for the timer ISR and a task for processing. The timer should be scheduled on a regular basis, and the interrupt handler should signal the processing task. To ensure that the timer is being triggered with the correct periodicity, pass the interrupt timing to the processing task.

ANSWER

Timer 0 was configured and enabled as a periodic timer with Period 1 second. This was done using the `TIMER_CFG_PERIODIC` API. The interrupt handler for Timer 0 was enabled. In the same manner, Timer 1 was configured and enabled as a periodic timer with Period 4 seconds. This timer is used as an absolute reference and to ensure that the timer 1 periodicity is correct. Task 1 is initialized and runs. It is signaled by the user-defined Interrupt Handler periodically, after every 1 second. Task 1 waits on the semaphore from the Interrupt Handler and is triggered after the timer runs out. The semaphore is released and the task is completed. This process repeats periodically. The Interrupt Handler thus handles Task1. The screenshot showing the execution of Q1 is as shown.



```
Waiting for Semaphore
Semaphore Received from Interrupt
Task got Semaphore after 4354512 Clock Cycle
Waiting for Semaphore
Semaphore Received from Interrupt
Task got Semaphore after 3995702 Clock Cycle
Waiting for Semaphore
Semaphore Received from Interrupt
Task got Semaphore after 3999958 Clock Cycle
Waiting for Semaphore
Semaphore Received from Interrupt
Task got Semaphore after 3999886 Clock Cycle
Waiting for Semaphore
Semaphore Received from Interrupt
Task got Semaphore after 4000096 Clock Cycle
```

Thus, the task is getting triggered periodically by Timer 0 ISR.

2.

Create a pair of tasks that signal each other. The first task performs some computation, signals the other task, and waits for a signal from that task. The second task repeats the same pattern so that they alternate. Each task should complete a defined amount of work, such as computing a specified number of Fibonacci values. Profile each task so that one task is executing for 10 ms and the other for 40 ms.

ANSWER

Two tasks – Task 1 and Task 2 were initialized. They signal each other using two semaphores. Task 1 runs for 10 milliseconds and Task 2 runs for 40 milliseconds. The two tasks complete a Fibonacci series. Task 1 has higher priority than Task 2. Hence, Task 1 gets released first.

The semaphores are initialized with value = 0. Task 1 is signaled in main(), the semaphore gets value = 1 and is acquired by Task 1. Task 1 then executes and runs the Fibonacci load for 10 ms.

After completion of Task 1 for 10 ms, Task 2 is signaled by giving the semaphore to Task 2 which then executes for 40 ms. After this, Task 2 signals Task 1, and this process repeats indefinitely.

Timer 0 is initialized as a free running timer to compute the execution times of both tasks (Task 1 and Task 2). The timer is started before the Fibonacci load, stopped and read after the load has finished executing.

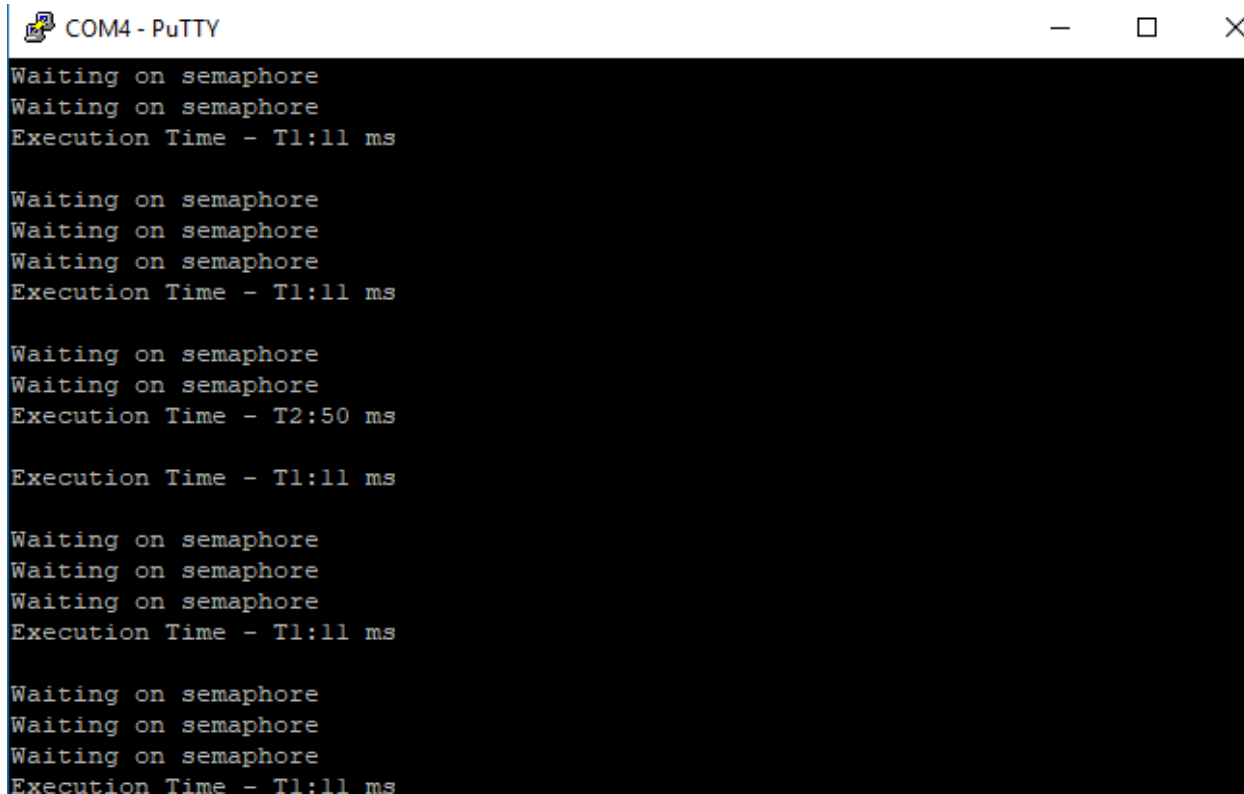
Thus, Task 1 runs for 10 ms, and then signals Task 2 which runs for 40 ms, which then signals Task 1 again, and this happens continuously.

```
Task1 Running
Load Time for Task1: 9 ms
Task2 Running
Load Time for Task2: 51 ms
Task1 Running
Load Time for Task1: 9 ms
Task2 Running
Load Time for Task2: 51 ms
Task1 Running
Load Time for Task1: 9 ms
Task2 Running
Load Time for Task2: 51 ms
Task1 Running
Load Time for Task1: 9 ms
Task2 Running
Load Time for Task2: 51 ms
Task1 Running
Load Time for Task1: 9 ms
Task2 Running
Load Time for Task2: 51 ms
```

3.

Modify the timer ISR to signal two tasks with different frequencies: one task every 30 ms and the other every 80 ms. Use your processing load from #2 to run 10 ms of processing on the 30-ms task and 40 ms of processing on the 80-ms task.

ANSWER

A screenshot of a terminal window titled "COM4 - PuTTY". The window has a black background with white text. The text shows a sequence of operations: "Waiting on semaphore", "Execution Time - T1:11 ms", and "Execution Time - T2:50 ms". This sequence repeats several times, indicating the execution of two tasks with different periods and execution times.

```
COM4 - PuTTY
Waiting on semaphore
Waiting on semaphore
Execution Time - T1:11 ms

Waiting on semaphore
Waiting on semaphore
Waiting on semaphore
Execution Time - T1:11 ms

Waiting on semaphore
Waiting on semaphore
Execution Time - T2:50 ms

Execution Time - T1:11 ms

Waiting on semaphore
Waiting on semaphore
Waiting on semaphore
Execution Time - T1:11 ms

Waiting on semaphore
Waiting on semaphore
Waiting on semaphore
Execution Time - T1:11 ms
```

Tasks runs at 30ms and 80ms while the tick time for the tasks is 10ms and 40ms respectively.

Software timers are initialized and configured as periodic timers with period 30ms and 80ms respectively.

Task 1 has a load of 10ms and Task 2 has a load of 40 ms which is generated by using the Fibonacci Synthetic Load function.

Task 1 runs for 10 ms every 30 ms, Task 2 runs 40 ms every 80 ms.

During the LCM multiple periods, both the tasks run together.