Indian Institute of Technology Bombay
CS684 Autumn Semester 2016

Scheduling in TI-RTOS

E.R.T.S. Lab

September 29, 2016

1 Lab Objective

- Analysis of time driven scheduling using the game console and TI-RTOS.
- Identifying tasks and implementing scheduling strategies.
- Understanding advantages of RTOS over sequential execution of code.

2 Pre-requisite

- Lab5: Interfacing GLCD.
- Lab6: Installing TI-RTOS.
- Basics of Real-time operating systems.
- Scheduling in Real-time operating systems.

3 Background Material

You will be give a test code (gameConsole.c) which interfaces all the available peripherals on the game console.

1. Four push buttons.
2. One shoot button with joystick.
3. Four leds.
4. Analog potentiometer (up-down direction of joystick).

5. One buzzer.
6. One graphics LCD.

The test code tests the board in the following way:

- Four push buttons are used to test the four Leds. When you **long press** the switch, one led corresponding to it with will get lit.
- The shoot button of the joystick is used to turn on the buzzer. When you **long press** the shoot button the buzzer turns on.
- The joystick (up-down) decides the delay between the two successive images on the GLCD. In our case its the cheetah animation. Its speed changes when you hold the joystick at either ends.

Observe the output of the code on the game console. Study the code and reason out why we need to long press any switch to observe the output.

4 Problem Statement

Write the RTOS version of the gameConsole.c code using TI-RTOS. The expected output is that you no longer need to long press the switches to observe the output. You should observe the LED output in real-time; without causing a lag on the GLCD. We will analyze how this can be done in the following sections.

5 Analysis

Here is how we can go about it:

- Let us say that we divide the entire code into three major tasks.
  - **Task 1:** Read the five switches(four push buttons plus shoot button) and update the leds and buzzer.
  - **Task 2:** Read the ADC value and update the delay to change the speed of animation.
  - **Task 3:** Update the image on the GLCD.
- For the purpose of analysis, let us assume that task 1 takes 20 ms to execute, task 2 takes 30 ms to execute and task 3 takes 100 ms to execute.
- Draw a time-line, if we were to implement this in sequential manner. X-axis will be the time in ms and Y-axis will be task that is being run. What will be range in time in which the switch presses will go undetected.
- Now, for implementing it in RTOS, we need to decide at what time you should schedule the tasks, with what period, so that the when you press the switch it should be detected without causing a lag on the GLCD screen. **Note:** TI-RTOS uses a preemptive scheduler.

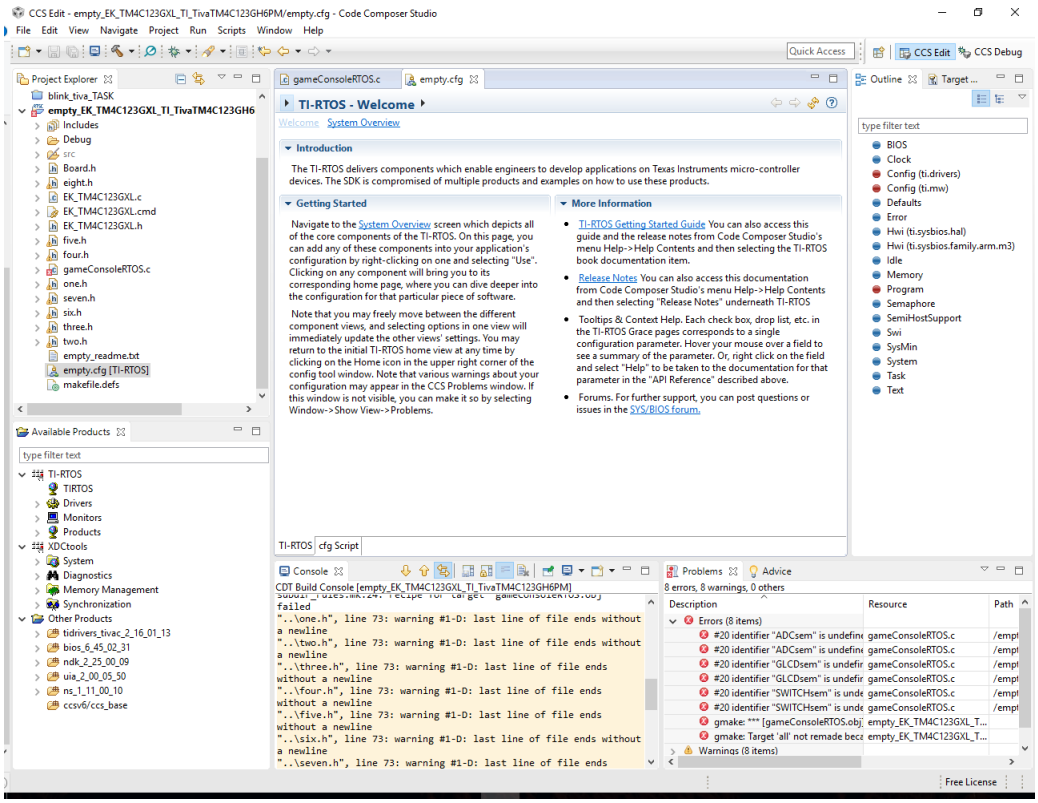
- As there are three tasks we can give them a priority. Decide a suitable priority for each of the tasks. Generally, a user expects that action of pressing the switch get instantaneously reflected on the output. In our case leds and buzzer. Keep this in mind while deciding the priority.
- We will create three tasks and three semaphores corresponding to the three tasks. After deciding the time when we should release the semaphores you can start writing the scheduling strategy.

6 Procedure

- Create a new project (non RTOS project) and include the gameConsole.c code in the project.
- Build and compile the code. You will need the image files that you used in Lab 5 *(Available on the course web page(part3))*. Study the code and observe output. Observe the behavior of the switches.
- Complete the tasks that has been described in the analysis sections based on your observations.
- Complete the analysis, you should have the scheduling strategy ready with you before proceeding further.
- Now to begin the RTOS experiment, go to projects ⇒ examples ⇒ RTOS for Tiva C ⇒ browse to the empty project and include it in the work-space.
- Replace the code in the empty.c file and with the code in the gameConsoleRTOS.c file. There are some missing parts in the code. Read through the code and complete it. It is related to the scheduling strategies that we have analyzed earlier.
- **Note:** If you get an error in the line

  ```
  #include "inc/hw_gpio.h" replace it with
  #include <inc/hw_gpio.h> and vice versa
  ```

  They seem to be the same but sometimes there are issues while porting from linux to windows and vice versa. If you do not get an error then ignore this step.
- Add the image files one.h, two.h ... to eight.h to the project. (Lab 5 part 3)
- There will be a empty.cfg file in your project. Open this file. You should be able to see the following windows.



Note(for linux): If you get an xdctools error while opening the config file. Right click: on your project ⇒ select properties ⇒ general properties ⇒ RTSC tab and select a different version of XDCtools.

- Once the GUI for the config file is open, first go to the outline window and select Hwi(ti.sysbios.hal). You have to add an Hwi for timer 2A. We will use this timer to trigger the tasks. Name the handle as **Timer_2A_INT** and ISR function as **Time_ISR** and interrupt number **39.** Keep rest of the settings as default.

- Go to task in the outline windows and create 3 tasks with the following details:

  - Handle: readADCtask Function: readADC
  - Handle: readSWITCHtask Function: readSwitch
  - Handle: updateGLCDtask Function: updateGLCD

  Priority for the three tasks has to be decided by you. Enter the suitable values in the priority field.

- Go to semaphore in the outline window and create three binary semaphore keeping rest of the settings to default with following details

  - Handle: ADCsem
  - Handle: GLCDsem

  - Handle: SWITCHsem

- Now read the code and fill the timer interrupt ISR depending upon your scheduling strategy. Find out the timer period from the timer init function. We have defined a variable called tickCount. This is incremented in the timer ISR. You can use this variable to decide when to release the semaphore. The syntax for the same is given in the comments section.

- Writing the scheduling strategy involves checking the tickCount variable and realsing the semaphore related to a particular task.

- Each task created and is ready to execute. The

  ```
  Semaphore_pend(ANY_SEMAPHORE, BIOS_WAIT_FOREVER);
  ```

  blocks the task. Until it gets a semaphore it will stay there forever.

- To release a semaphore use

  ```
  Semaphore_post(ANY_SEMAPHORE);
  ```

7 Demo and Submissions

This exercise is a lab cum assignment.

You have to show the output of the test code using RTOS where the leds and buzzer will respond to the switches in real-time without any lag seen on the animation on the GLCD.

We will accept the assignment part on Moodle. The assignment should contain:

- The time-line of the three tasks executed sequentially (ref analysis section).

- Mention the priority that you used for the three tasks. Justify why you assigned a particular priority to a particular task.

- Explain your implementation and scheduling strategy in the document in detail.

- Also, mention if the switches respond in real-time if you implement the test code using RTOS. Was there any lag observed on the GLCD ?

Pleas upload a single PDF on Moodle. You can work in a team for the lab part but upload individual documents on Moodle for the assignment part.