CPE 490 Lab 10 S14

## Goals

1. To get a RTOS up and running and to create a few tasks with different priority.

## Overview

This lab is a repeat of Lab 9 but this time we will use an honest to goodness RTOS instead of the poor man’s version we used in LAB 9. The RTOS we will be using is named freeRTOS. Go to freeRTOS.org and lecture notes to see details. The porting of the RTOS even from the instructions on the web is more than you can get done in a lab so the code in main() will be provided as well as nearly empty function calls. You will write the already named functions that will be the core of the tasks. Since the tasks will be the same as LAB 9 you should have most of the code that you need just modified within the scheme of the RTOS.

## Design

### Program Overview

The program will do the exact same function as the program described in Lab 9. You will write three functions that will be used to create tasks; two will be the same priority and the third will be a lower priority.

### Setting up the program

Copy the zip file from blackboard and install the files in a place of your choice on your hard drive. Open this project by using the File menu and then selecting Open\_Project. The Project Folder can be found in where you copied the zip file and then lab10 Post\Demo\dsPIC\_MPLAB\RTOSDemo\_dsPIC.x. Open this workspace.

Next compile the code, it should compile without error. If it can’t find the compiler, or linker just take the default suggestion.

### Writing code

You will write three functions, the first two are used when making a task in main() using the API xTaskCreate() with a priority of 2, as seen in the given main() code. The third function is a task that runs at a lower priority 1.

All three tasks must be now written to be endless loops that never try to return, a for( ; ; ) loop is provided in the main.c file for this purpose.

Since we have multiple priority levels every task must use an API delay functions vTaskDelay() or vTaskDelayUntil () for a periodic task like a RTOS (see lecture 16 if you don’t remember this). Even the lower priority WriteLCD task must relinquish the processor through this API in order that the idle function can run. Certain housekeeping functions are done in the idle task so it can’t be starved.

Since both the RTC and the counting tasks might want to publish a string to the LCD, it makes sense to make a queue that either of the tasks can write to, and the display code can read from. A queue is created for this purpose in main() with the following:

LCDDisplayinfo = xQueueCreate(2,16);

This queue will have a length of 2 and each chunk will hold a 16 character string.

Details on the 3 functions are given next:

**static void ButtonPushCounter( void \*pvParameters);**

This function will scan the button switches as before in lab 9. It will increment the counter and it will also set the global flag RTCFlag if the RTC is to be displayed on the LCD. Remember from lecture we must write this function so it never returns. So once we declare variables and do any initialization the code should go into and endless loop. Variable should not be defined as static, as discussed in the lecture. Once the scan and any other coding related to the scanning is done; you will need to block the task for running for a set amount of time. To do this you will make the API call using vTaskDelay(). The prototype is:

void vTaskDelay( portTickType xTicksToDelay );

The value xTicksToDelay is the number of tick counts the task should be blocked. The tick counter will have a period of 1 ms, so just put in the number of milliseconds you would like to be blocked.

Reviewing lab 9: The inputs will be scanned every 10 ms until a push button is activated. The switch must be released before any new switch scanning can be done. To avoid switch bounce once a button is pressed the state of the input switches will not be scanned for 100 ms and then if released the scanning will resume to every 10 ms.

Most likely you will want to write a string to the queue from this task. The following is given:

xQueueSendToBack(LCDDisplayinfo, LCDDisplay, xTicksToWait);

Where:

* LCDDisplayinfo is a global variable given to you in the code that identifies the queue.
* LCDDisplay is the name of an array of 16 characters that you would like to put into the queue.
* XTicksToWait is an integer or can be a constant that represents the time in milliseconds to block waiting for the queue to open up.

**void RTCTimer(void \*pvParameters);**

This function will keep track of the RTC and load LCDDisplay with correct RTC string. This function must never return, like the previous function. When it gets done with updating and loading the queue (see statement above for an example) it should block itself. Since this is a periodic task with exact timing we need to ensure that the time it takes to get to the delay blocking API function does not add time to when it should be unblocked. To do this use the vTaskDelayUntil() function. Partial code in using this function has been given to you in the code download.

Since the resolution of the RTC is ½ a second, this task can block itself for 500 ms.

**void WriteLCD(void \*pVParameters);**

This task should print out one character at a time then block itself. Consulting the LCD controller data sheet we see that the longest time the controller ever needs to complete a data write or a command is 1.64 ms so make the delay time block it for 2 ms. The vTaskDelay() API will work well.

In order to get data from the queue the task will have use the following API:

NewString = xQueueReceive( LCDDisplayinfo,LCDDisplayString,10);

Where the queue is identified as LCDDisplayinfo, a local string of size 16 character array is called LCDDisplayString, and the task will stay blocked 10 ms waiting for data to be placed in it. You can examine NewString to see if a new data was copied out of the queue.

When you are finished demonstrate the code to an instructor:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_signature