Project 5 – Create a Custom Preemptive, Multitasking Operating System

Due: All Documents Due Before 5:00pm on Friday, April 10, 2020 (no changes after this date)
Demo/Presentations During Lab Period on April 16, 2020

Introduction

In this individual project, you will create a custom preemptive, multitasking operating system (PMOS) capable of running at least three tasks 'simultaneously.' Your PMOS will include the bare essentials of a kernel; full context switches between tasks and guaranteed processor time for tasks (scheduler). Your PMOS must also implement at least one of the following features: a) the ability to add new tasks on-the-fly (a task calls a new task) and to remove tasks when completed, or b) task priority-level management. You will be given code for one of your tasks (see Task2.c), a second task will be to count every second and display the count on the LaunchPad's LCD, and the third task will be for you to create. You may create additional tasks as needed to demonstrate the features of your PMOS. Finally, you will provide a short instruction manual for using your PMOS (give your PMOS a unique and creative name!).

In this project you are expected to use:

- MSP430FR6989 microcontroller
- LCD and other hardware on your LaunchPad
- The given LaunchPad LCD function library

Note that while your tasks will have timing constraints that have to be met, I do not expect you to include the time management features in the operating system that would make this a real-time operating system. Also note that this project does not require you to use the low power modes. If you can make this project work while also using low power modes within and/or between the tasks you will receive a significant bonus (but this does not replace any of the required PMOS features listed).

Description

In this project you will create and demonstrate a very basic preemptive, multitasking operating system (PMOS). Many of the elements will be discussed in class, but it will ultimately be up to you to design and implement your PMOS. This will not be a simple project. However, the operation of a PMOS should provide you with a strong understanding of how computers actually operate and should give you a very good talking point on your resume.

The main goal is to implement an PMOS task scheduler that performs full context switching and guarantees run time for each task. Your PMOS must also implement at least one of the following features: a) the ability to add new tasks on-the-fly (a tasks calls a new task) and to remove tasks when completed, or b) task priority-level management. It is up to you how to implement your PMOS and which additional feature(s) you wish to implement. Then you will use and create at least three tasks to demo the operation and features of your PMOS. The three required tasks are:

Task 1: Create a task (consisting of one or more functions) that implements a count-up timer that, once a second, increments an integer beginning at zero, and then displays the number on the LaunchPad's LCD (use the **myLCD_displayNumber** function from the **myLcd.c** library). In addition, when the P1.1 button is pressed this task will display on the LCD the words "START OVER" (stopping the display of the count) and then reset the count-up timer to zero and begin counting and displaying the numbers again each second.

Task 2 (given to you as **TASK2.c**)**:** This task, using timer TA2, outputs a 1 Hz, 50% duty cycle to P9.7 (the Green LED). In addition, every three seconds this task reads a global variable, **unsigned int HoldGreenLED**, and will override the P9.7 output to turn on the Green LED for the number of seconds given by the integer value contained in **HoldGreenLED**. This global variable is a non-binary semaphore that your **Task 3** will be setting. After the Green LED has been on for the number of seconds given in the semaphore, the P9.7 output will return to the 1 Hz 50% duty cycle signal.

Task 3: This last required task is for you to create with only two requirements. First, every 10 seconds you must place a new integer between the values of 0 and 25 into the non-binary semaphore **HoldGreenLED**. You may use any method you wish to create the integer value including random generation or user input using the buttons (remember that **Task 1** uses the P1.1 button). Finally, this task must do "something interesting" with the P1.0 Red LED. Otherwise, be as creative as you wish with this task.

You may create additional tasks if you wish to demonstrate additional functionality of your PMOS.

Successful implementation of this project will require a mixing of both C and assembly code. The majority of code can be written in C, however, due to the low-level nature and memory constraints, you will find that it is necessary to implement some functions in assembly (or utilize the inline **asm()** function. See your C compiler user's guide). You may also disable the watchdog timer for this project. Your main demo program might look similar to the following code:

main.c

```c
//**************************************************************************
// Description and Authorship
//**************************************************************************
//**************************************************************************
// Includes
//**************************************************************************
#include <msp430.h>
#include "MyRTOS.h"

//**************************************************************************
//  Definitions and Global Variables
//**************************************************************************
char TaskCounter = 0;    // keep track of number of tasks running
char MaxTasks = 10;      // max number of tasks allowed to run at once
                         // these global variables would be better to declare
                         // in your MyRTOS.h or MtRTOS.c files

//**************************************************************************
//  Function Prototypes
//**************************************************************************
int task1(void);         // declared as an int because the task will return
int task2(void);         // a zero when finished
int task3(void);

// the following functions would be better to declare in MyRTOS.h or MyRTOS.c
void RTOSsetup(void);    // global setup like stop watchdog and turn on GPIO
void RTOSinitTask(void (*pFun)(void)); // initialize tasks (create TCB)
int RTOSrun(void);       // main RTOS scheduler program: if it returns a
                         // zero then all tasks are complete

//**************************************************************************
// Main Program
//**************************************************************************
void main(void)
{
    char error1;         // variable to check if all tasks are finished running

    // global setup instructions such as stop watchdog timer and turn on GPIO
    RTOSsetup()

    // Initialize the default tasks to run
    RTOSinitTask(task1);    // initialize task1 (create task control block)
    RTOSinitTask(task2);    // initialize task2 (create task control block)
    RTOSinitTask(task2);    // initialize task3 (create task control block)

    error1 = RTOSrun();     // main RTOS scheduler program: if it returns a
                            // zero then all tasks are complete
                            // error and do a PUC on any other return

    if(error1 == 0)  // if all tasks have finished
    {
        // go to low power mode code here
    }

    // otherwise ERROR! put code here to do a PUC (soft reset of the microcontroller)

} // end main()
```

The code shows the three tasks to be implemented, a **RTOSsetup** function to take care of turning off the watchdog timer, turning on the GPIO and possible other things, a **RTOSinitTask** function that is passed a pointer to the Task function and will create the Task Control Block for the Task function, and finally the **RTOSrun** function which contains/calls the PMOS scheduler.

Note that the non-binary semaphore (global variable) **HoldGreenLED**, is not shown in the code above.

Demonstration/Presentation (60%)

On April 16th during your normal lab period, you will be scheduled to demonstrate and present your project to your instructor or TA in a Google Meet (or another online video tool that your instructor may utilize). You will have 15 minutes in an interactive online setting to explain and demonstrate the operation of your project. Be prepared with the project running, the code ready to display, and a few slides showing flowcharts, state diagrams, code snippets, or other figures that demonstrate the functionality of your project and the algorithms used (know how to present your screen in the online tool and how to switch between the camera showing your LaunchPad and different windows on your computer). Remember that in this one-on-one interactive session, your audience is supposed to be a semi-nontechnical manager who is making the decision on continuing funding for your project. Interactive means that the manager will be asking questions and making comments during the demo/presentation. You will receive marks for the required functionality, BONUS POINTS for any optional or extra features you have implemented, and you also will be scored on how convincing your one-on-one demo/presentation is for continued funding of the project.

The following are the Minimum Functionality Requirements for the project that is worth **40%**:

1. Implement a PMOS that:
   a) uses full context switching; and
   b) guarantees that each task gets some run time.
2. Implement at least one of the following features in your RTOS:
   a) the ability to add new tasks on-the-fly (a task calls a new task) and to remove tasks when completed;
   b) task priority-level management.
3. Implement the three required tasks with the specifications as described above.
   a) Task 1:
   b) Task 2: (Given to you).
   c) Task 3:
4. Create a demo program that demonstrates the features and functionality of your RTOS by running your three (or more) tasks.

The other **20%** of your Demo/Presentation score (40% + 20% = 60%) will be from how convincing, how well organized, and how professionally you presented your project in the one-on-one interactive session with the supposed manager. I will provide some more metrics on this next week, but they will be similar (but pared down) to the presentation metrics from the previous projects.

Please use a computer drawing program to create flowcharts, state diagrams, and other graphic aids for professional looking results both for your Demo/Presentation and for your documentation. For flowcharts, state diagrams, and other tree, graph, or bubble diagrams, I highly recommend draw.io (www.draw.io) which is a free to use browser-based application. Please present flowcharts state diagrams and other graphics as appropriate and make sure they can easily be read. Be proud of your work and show it off!

Documentation (40%)

For this project you must submit your fully documented code along with a matching report level flowchart(s) or state transition diagram(s). You must also submit any slides, diagrams, flowcharts, code snippets, and any other documentation you used in your demo/presentation. Your need to submit a completed work hour log sheet and you need to submit your user's manual. These are described in more detail below.

The final version of these documents must be submitted to Blackboard before 5 pm on April 10th. Remember that your demo/presentation is not until the following Thursday and these documents cannot be modified after the 10th. Therefore, make sure that you have prepared everything that you need for your demo/presentation. It would be a good idea to actually practice your interactive one-on-one, perhaps with another student to help ensure that it will go smoothly and that you have not forgotten any needed documentation or figures which your manager may ask for.

Since this is Valpo, at the end of each submitted document you must include the Honor Code, your names, and your signatures. The Honor Code and your names should be typed and for your signature you may use a script font instead of importing your signature or using a verified electronic signature.

Code

Your fully documented code must include a header comment section for each standalone code section. This header comment section must include a description of what the code does, the author's name(s), the creation date, the version and last modification date, and the purpose of the code (e.g., ECE-422, Project 1). By standalone code section, I mean any of your code functions in separate files or included previously written code functions in your main.c file. These reused code functions must contain their own header comment sections with the required authorship, date, version, and purpose information. If you use code functions supplied by your instructor or a third party it must be noted in the comment header section and if you modify such code that also must be noted along with revision dates.

Your code must be easily readable with good whitespace and separated into logically arranged sections with section header comments. The code must be commented such that a non-technical manager can read the comments and understand what the code is doing without having to actually understand the code.

Please submit the actual .c and .h files (find them in your CCS folder) and not a printout that contains the line numbers. Yes, on Blackboard the spacing and your carefully aligned code may not display properly, but when placed back into CCS it will look as you wrote it.

Flowcharts/State Transition Diagrams

As your projects get more complex, it may be more useful to show your program logic in state transition diagrams rather than flowcharts. This will be your choice on how to best display the logical operation of your code to your management team (i.e., instructor). The matching report level flowchart or state transition diagram should be a direct logical match to your submitted working code and must be at a level such that a non-technical manager can follow the logic and algorithms that you used to implement your solution. At the top of your flowchart/state transition diagram document there should be authorship information similar to that in your code. Flowcharts/state transition diagrams must be submitted as PDF documents.

Instruction Manual

You must provide a short instruction manual that clearly describes to a user/developer how to use all the functionality of your PMOS. Your user manual should be clear and complete and should include simple code examples. This should be submitted as a Word (.doc, .docx) or a text-based PDF document.

Any Additional Slides and Documents from your Demo/Presentation

Include any slides you used in your presentation including code snippets and zoomed in flowcharts or state diagrams. Any slides used in the demo/presentation must be submitted as a .ppt or .pptx file if you used PowerPoint, and as a PDF file if you used other presentation software (unless it is a standalone executable).

Work Log

You must fill in your worklog as you go, rather than wait till the end. Include brainstorming, planning, and debugging time. Make the entries descriptive of what you did on the project and round to nearest quarter hour. This should be submitted as a Word (.doc, .docx) or a text-based PDF document.