EE475 Lab #6 Fall 2004

Extended Multitasking

This week you will create another non-preemptive priority multitasking system. The system will have only a few tasks to do, but it will illustrate several additional features of a simple kernel system.

Preliminaries

1. Be sure to have a copy of Lab #5 and the Cady M68HC12 book on hand as a reference for any questions on the port bit assignments, etc.
2. Make a temporary local folder for your work:
   `c:\EEClasses\EE475\tempxxx` .
3. Launch Code Warrior and make a new project file for this week.
4. Replace the `main.c` file with your program from Lab #5 or some other previous lab example.

Exercise #1: Set Up Several Tasks

To get started with the simple kernel, download the C program framework file (`lab6.framework1.txt`) from the course website. *You will need to edit the framework file to add your own code.*

The framework file defines an 8-bit variable called `interrupt_pattern`:

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
unused	unused	unused	unused	(see Ex. #3)	IRQ	RTI 1	RTI 0

a) The provided function `RTI_handler()` sets bits 0 and 1 of `interrupt_pattern` on a periodic basis. *You will need to install a pointer to this function in the RTI location of the vector table.*

b) You need to write a function `IRQ_handler()` that sets bit 2 of `interrupt_pattern` whenever the IRQ button is pressed. You will need to install this function pointer in the vector table. Also, *set IRQ to be edge triggered.*

c) Write code for four tasks: `func0()`, `func1()`, `func2()`, and `idle()`.
`func0()` must flip the state (0→1 or 1→0) of pin 0 in `PORTT`.
`func1()` must flip the state of pin 1 in `PORTT`.
`func2()` must flip the state of LED 0.
`idle()` must increment a global variable (`idle_count`) to track how many times it runs.

d) Write a temporary `main()` function to test your four routines. *For now, do not use interrupts*, just put all four routines in a loop with a delay so that the state toggling happens about once per second. Observe the `PORTT` pins using the oscilloscope, and observe the LED directly.

Exercise #2: Run Functions Using a "Task List"

Edit your `main()` program to declare an array named `func_table[]` containing three pointers to functions with no return value and no arguments. Set the three function pointers to `func0`, `func1`, and `func2`. This is your rudimentary task list.

Alter your test loop from Exercise #1 above so that `func0()`, `func1()`, and `func2()` are executed as before without interrupts, but now via the pointers stored in `func_table[]`.

→ Demonstrate the proper operation of your functions and* `func_table` *access for the instructor. Determine where* `idle_count` *is stored in memory and use the monitor routines to display it.

Exercise #3: Task Control "Kernel" Setup

The simple kernel concept used in this lab has the following properties:

- If toggle switch 7 is 'off', stop execution and exit.
- Task 0 is the highest priority, Task 1 is the next lower priority, etc.
- Task 0 is "ready" if toggle switch 0 is 'on' AND bit 0 of `interrupt_pattern` is '1'
- Task 1 is "ready" if toggle switch 1 is 'on' AND bit 1 of `interrupt_pattern` is '1'
- Task 2 is "ready" if toggle switch 2 is 'on' AND bit 2 of `interrupt_pattern` is '1'
- If none of the three tasks is ready to run, then run the `idle()` function.

Assign your `func0`, `func1`, and `func2` to be Task 0, Task 1, and Task 2, respectively. One way to implement these properties is given in the code fragment shown below.

```
#define NUM_TASKS 3

while(!(PORTAD&0x80)) /* exit if toggle switch 7 is 'off' */
{
       i=0;  /* start with task 0 */
       while(i<NUM_TASKS)
       {
              /* A task can run if its toggle switch is on AND the corresponding
               * bit is set in the interrupt_pattern variable.
               * Task priority order is 0, 1, 2, … then idle.
               * Stay in this while-loop until no task is currently 'ready'.
               */
              if ( test involving switches, task number i, and interrupt_pattern goes here )
              {
                     /* YOUR CODE GOES HERE:  execute function pointed
                      * at by func_table[i], and then
                      * clear the ith bit in interrupt_pattern
                      */
                     /* Reset i to always check task 0 first (highest priority)
                      * after any task is run.
                      */
                     i=0;
              }
              else i++; /* increment task number if current task not ready */
       }/* end while i */
/* This point is reached when above NUM_TASKS tasks currently unable to run.
 */
  idle();
} /* end while switch 7 */
```

Using this code fragment as a model, write the required instructions so that the kernel calls your assigned functions. Verify that the toggle switches control execution of the three tasks. Check to see that your `idle()` function gets called and that the idle count global variable is being incremented.

Exercise #4: Add another task

Finally, modify your program to include another task (task 3). Have task 3 toggle LED 1.

The new task (task 3) should be managed like the others: ready to run if toggle switch 3 is 'on' AND bit 3 of `interrupt_pattern` is set. However, instead of setting the `interrupt_pattern` bit in the ISR, modify your `func2()` routine so that *it* sets bit 3 of `interrupt_pattern` *every other time* that IRQ is pressed. In other words, your new task 3 will be ready to run only if task 2 enables it.

→ ***Demonstrate the four task + idle system for the instructor.***

Instructor Verification Sheet
Lab #6 Fall 2004

Student Name: _____

	Instructor Signature	Date
Ex. #2 Tasks executing using lookup pointer table		
Ex. #4 Four tasks executing with RTI/IRQ/switch controls		

Lab Report

The lab report is to be written up in the Memo format. Be sure to put the *lab number* in the Memo header along with your name and date. For each exercise, answer the given questions and demonstrate your understanding of the exercise. Include **commented** file excerpts and this instructor verification sheet to get credit for the lab.

→ This lab report is due the beginning of the lab period in one week.