Department of Electrical and Computer Engineering

ECE 5710/6710 Lab 6

Title: A Preemptive Priority-Driven Scheduler

Objective: The student should be able to write code to implement a fixed priority, preemptive scheduler.

Parts: 1-EFM32GG Evaluation Board

Software: Silicon Laboratories Simplicity Studio.

Preparation: Write the title and a short description of this laboratory exercise in your lab book. Make sure the page is numbered and make a corresponding entry in the table of contents.

You are to schedule the following four tasks (which are the same as the tasks in lab 5):

Task	Period/Relative Deadline	Execution Time (e_i)
A	3 ms	0.5 ms
B	5 ms	0.9 ms
C	6 ms	1.1 ms
D	11 ms	2.4 ms

The code that executes the jobs for each of these tasks is provided in the file tasks5.o. To execute a job for a particular task, you need only to call task_N() where N ∈ {A, B, C, D}. Note that all of these functions are preemptable.

You are to write code that releases these tasks periodically. To do so you will need a global tick counter that is incremented once each millisecond, and you will need to allocate one (32-bit) word in each task control block (TCB) to hold the next release time (tick) for each task. You will then write a scheduler that selects the highest priority task that is also released (i.e. the global millisecond count is greater than or equal to the release time for that task). If no task is ready the idle task (TCB[0]) should be scheduled.

You will have to decide how to assign priorities to tasks. One common approach is to add a new member to the task control block that indicates task priority. If you take this approach you will need to modify CreateTask() to initialize the new member. Another common approach is to assign priority by virtue of a task's position in the TCB array.

It is recommended that you copy the code you developed for the round-robin scheduler in Lab 4. Modify the copy to configure the SysTick timer to interrupt once each millisecond, then modify the SysTick handler so it increments a global system tick variable. The code snippet below may be helpful:

```
SysTick_Handler:
    push    {r4-r11, lr}
    ldr     r6,=SystemTick   // r6 is address of system tick
    ldr     r7,[r6]          // r7 is current tick
    add     r7,r7,#1         // increment tick count...
    str     r7,[r6]          // ...and store it
    …
```

You will, of course, want to make sure your system tick starts at 0.

Next, create a function that behaves something like vTaskDelayUntil() in FreeRTOS. The most straightforward approach is for your function to (a) take a parameter, (b) store it in the current task's TCB and (c) invoke the scheduler. It is not difficult to invoke the scheduler from a task if you use the Yield() function and call the scheduler from inside SVC_Handler.

Each task loop might now look something like this:

```
void task_loop_N(void)
{   int release_time = 10; // release all tasks at t = 10
    while (1)
    {
        your_wait_function(release_time);
        task_N();
        release_time += period;
    }
}
```

An astute programmer will be justifiably concerned that the global tick value will eventually "roll over" and become negative, which will mess everything up (much as Y2K was supposed to do). While the concern is valid, it will take about 25 days for that to happen and hopefully you will have moved onto the next lab exercise by then.

Finally, your main function should create the 4 periodic tasks then go into an idle loop where it continually increments a loop count variable. Make sure optimization is off so you can debug and see this variable.

Build your code and see to it there are no errors. Don't forget to copy tasks5.o and add "${ProjDirPath}/tasks5.o" to the linker properties.

Procedure: Launch the debugger to download your code. Run it and verify both amber LEDs light up and remain lit. If not, the debugging tools that you used in previous labs are still available (e.g. get_time, early_task, etc.).

Stop your code and note the value of your loop count. Restart and stop it again to verify the loop count is changing. When your program works, demonstrate it to your lab instructor.

Print a copy of your code and affix it into your lab book. Write a short summary or conclusion and submit your lab book to your lab instructor for grading. (Remember to initial and date!)

Points will be assigned according to the rubric below:

Criterion	Points
Lab book properly bound and kept in ink	1
Lab book contains a title and short description	1
Each page initialed and dated, large areas crossed out	1
Pages are legible with no obliterations	1
Lab book contains scheduler and context switch code	2
Program works	4

Late work is penalized 2 points (20%).