Department of Electrical and Computer Engineering

ECE 5710/6710 Lab 9

Title: Implementation of Semaphores

Objective: The student should obtain experience writing code to create, take and give semaphores.

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 write an application based on your work in Lab 6 to schedule the following four tasks (which are the same as in Lab 8).

Task	Period	Relative Deadline	Execution Time (e_i)
A	3 ms	3 ms	0.5 ms
B	125 ms	10 ms	1.0 ms
C	29 ms	29 ms	2.0 ms
D	49 ms	49 ms	7.0 ms

The code that executes the jobs for each of these tasks is provided in the Lab 8 assignment on Canvas (file tasks8.o). The tasks and what you are to do with their results are identical to Lab 8, but you will be using your own scheduler and your own semaphores.

As a reminder, the jobs in tasks C and D must be executed in one critical section, and function calls to the LCD driver must be executed in another (once scheduling begins). These critical sections must be guarded using two separate semaphores.

You will also need to modify the include path and link (or copy) the files segmentlcd.c and em_lcd.c just as you did before. When you are finished, the LCD should behave exactly as it did in Lab 8.

For this laboratory exercise, you will need to create your own semaphores. You can implement them as a structure or class, you may have an array of them and index them with an integer, or you may simply use an integer whose address serves as a unique identifier. Whatever works for you will be acceptable.

There are some provisos…

1. You must be able to create multiple, independent semaphores.
2. When an attempt to take a semaphore fails, the task must yield the processor.
3. When a semaphore is given, the task must yield if and only if (a) there is another task waiting for the semaphore and (b) that task has a higher priority.
4. Your semaphore code should not yield the processor in any other circumstance.

You will need to modify your scheduler so that tasks that are blocked on a semaphore are not scheduled. Note that tasks may only be blocked by one semaphore at a time, so one extra field in the TCB should suffice.

When implementing semaphores, be very careful not to allow pre-emption at certain key times. For example, consider the pseudocode below:

```
void take_semaphore(semaphore *s)
{
    if (s->count > 0) // check if semaphore is available
    {
        s->count--; // it is! So take it
        …
```

If this code is preempted after the test is made but before the count is decremented, it is possible that this task and another will be in the critical section simultaneously (and the semaphore count may be less than zero). To avoid this problem, you can disable interrupts (making a small, non-preemptable critical section) within your code:

```
void take_semaphore(semaphore *s)
{
    __disable_irq();
    if (s->count > 0) // check if semaphore is available
    {
        s->count--; // it is! So take it
        __enable_irq();
        …
```

You may assume that interrupts are enabled when your semaphore functions are called; just make sure they are still enabled when they return (or call Yield()).

Build your code and see to it there are no errors.

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 earlier lab exercises are still available (e.g. get_time, early_task, etc.).

Verify that the progress ring on the LCD rotates smoothly, and that the numbers on the LCD are advancing without flicker.

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 main program and context switch code	2
Program works	4

Late work is penalized 2 points (20%).