Department of Electrical and Computer Engineering

ECE 5710/6710 Lab 8

Title: Using Semaphores to Establish Critical Sections

Objective: The student should be able to use semaphores to guard critical sections against preemption.

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 a FreeRTOS application to schedule the following four tasks:

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 for each of these tasks is provided in the file tasks8.o. As usual, to execute a job for a particular task, you need only call its function. This time, some of the functions have parameters that you will need to provide:

```
extern void task_A(void), task_B(void), task_C(char *), task_D(int *);
```

The jobs in tasks C and D are mutually exclusive, which means that you must not preempt one in order to execute the other. Both may be preempted by tasks A or B, however. Your code will need to create a semaphore and use it to establish critical sections around the calls to taskC() and task_D().

You will also be using the LCD to output the results from some jobs, and the LCD driver provided by Silicon Labs is not thread-safe. You must therefore treat the driver as a resource and call its functions only within critical sections guarded by a <u>different</u> semaphore.

Create a FreeRTOS project as you did in Lab 5 or Lab 7. Use either technique to schedule the tasks at the periods specified above. Employ a deadline monotonic priority assignment.

Your project will need to link some additional files from the SDK in order to use the LCD. First, you will need to find the header files segmentlcd.h and segmentlcdconfig.h in your SDK and add their folders to the include path (refer to Lab 5 to see how). For the current SDK, those folders are:

${StudioSdkPath}/hardware/kit/common/drivers
${StudioSdkPath}/hardware/kit/EFM32GG_STk3700/config

Next you will have to find (and navigate to) the folders that contain the files segmentlcd.c and em_lcd.c. In the current SDK they can be found in:

{StudioSdkPath}/hardware/kit/common/drivers
{StudioSdkPath}/platform/emlib/src

Drag these files into your project folder WITHIN THE IDE. You will have a choice to link to these files or copy them. You will not be changing these files so either choice should be fine.

Add the following line to your main C file:

```
#include "segmentlcd.h"
```

The LCD driver must be initialized before you can use it. To do so, call the following function before scheduling any tasks:

```
SegmentLCD_Init(false);
```

The functions that output numbers and strings on the LCD take longer the first time they are called, so you should make the following additional function calls as part of the initialization, for example:

```
SegmentLCD_Number(0);
SegmentLCD_Write("HELLO");
```

For this laboratory exercise, you are required to advance the progress ring each time a job from task B is executed.. To do that, declare a variable (e.g. prog) then insert the following code somewhere after calling task_B():

```
    SegmentLCD_ARing(prog,0); // turn off previous segment
    prog = (prog +1) & 7;
    SegmentLCD_ARing(prog,1); // turn on next segment
```

You are also required to output the values generated in tasks C and D to the LCD. The function task_C takes a pointer to an array of 8 or more chars and fills it with an ASCII string to output. After calling task_C, output the string to the LCD as illustrated by the code fragments below:

```
char s[8];
…
task_C(s);
…
SegmentLCD_Write(s);
```

The function task_D also takes a pointer, but this time it is a pointer to an integer. Task_D() fills this integer with the number to output on the LCD as illustrated below:

```
int n;
…
task_D(&n);
…
SegmentLCD_Number(n);
```

Add code to create semaphores and to take and release them before and after each critical section. (Caveat: in FreeRTOS semaphores are created empty. You will need to free (give) each semaphore before it can be used to guard a critical section.)

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. 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 the software listing	1
Program works	5

Late work is penalized 2 points (20%).