Department of Engineering

EE 4710 Lab 5

Title: Using Semaphores

Objective: The student should become acquainted with the operation of semaphores and how they can be used to release tasks and control access to resources in a real-time operating system.

Parts: 1-C8051FX20-TB Evaluation Board
1-USB Debug Adapter
1-DB-9 Serial cable (USB adapter cable is also ok)

Software: Silicon Laboratories IDE version 3.50.00 or greater. Keil compiler.

Background: The time delays we learned about in Lab 4 accurately release periodic tasks only if (a) the execution time of the task is smaller than a timer tick and (b) the task is not blocked from executing by higher priority tasks. In practice, this does not happen frequently.

One solution to this problem is to use a high priority task whose only function is to signal lower priority tasks when they have been released.

Procedure: Write the title and a short description of this lab in your lab book. Make sure the page is numbered and make an entry in the table of contents for this lab.

Write a program using the salvo operating system (you may need to go back and consult Lab 2). Use salvocfg to configure the system for 3 tasks and 2 events. (Otherwise, use the same configuration you used in Lab 2).

In order for salvo to handle delays, you must call OSTimer() once per timer tick (usually from an interrupt service routine). Your program, therefore, should set up one of the 8051 timers and include an interrupt service routine that calls OSTimer. For example:

```
void timer2(void) interrupt 5
{
  TF2 = 0;
  OSTimer();
}
```

You will need to enable the 22.1184MHz oscillator and configure a timer to interrupt 100 times per second (for Timer 2, this means loading RCAP2 with -18432). Once you have done so, write a program with a single periodic task that (a) turns the LED (P1.6) off for 20ms (2 ticks) and turns the LED on for 480ms (48 ticks). Your task must use the function OS_Delay() whenever delays are needed. (OS_Delay() behaves much like sleep() does from Lab 4.)

Compile your program to make sure it has no errors. If possible, download and run your program as well. Verify the LED is mostly on but blinks off twice per second.

Your program will now be required to send the string “No, you can’t” to the terminal whenever the LED is turned off and “Yes, I can” whenever the LED is turned on. You may not do this with a single task. Instead, the task you have already written will control the release of two other tasks that actually send the strings.

One way to control the release of a task is to have the task “take” a semaphore. If the semaphore is 0 (empty, i.e. not released), the task blocks and waits until another task (or interrupt service routine) “gives” the semaphore. Once the semaphore is given (i.e. released), the task becomes ready to run and can be scheduled by the scheduler at its earliest convenience. (When the task completes, it awaits the next release by taking the semaphore again.)

To use semaphores, you must first create them (and initialize them to be empty). In salvo, this is best accomplished first by defining pointers to their event control blocks (ECBs), then by initializing them with OSCreateBinSem() as shown below:

```
#define SEM_RELEASE_1  OSECBP(1)
#define SEM_RELEASE_2  OSECBP(2)
…
//
// put this in main() before enabling interrupts
//
   OSCreateBinSem(SEM_RELEASE_1, 0);
   OSCreateBinSem(SEM_RELEASE_2, 0);
```

To use the semaphores, go to the point in the code where the LED is turned off and add the code:

```
     OSSignalBinSem(SEM_RELEASE_1);
```

Do the same for the code where the LED is turned on (except signal the other semaphore). Create two new tasks. Give each the same priority, but a priority lower than the first task. Each new task should consist of an infinite loop. In each loop write code that (a) waits for the semaphore, (b) outputs a string, and (c) repeats. (See code, below.)

```
for ( ; ; )
{     static char code *s;
      OS_WaitBinSem(SEM_RELEASE_1, OSNO_TIMEOUT, label_goes_here);
      // execute the job here
      s = "No you can't!\r\n";
      while ( *s )
      {
        TI0 = 0;
        SBUF0 = *s++;
        while ( !TI0 ) OS_Yield(another_label_goes_here);
      }
```

In salvo, all tasks use the same stack, so it is necessary that all context switches (e.g. OS_Yield()) occur in the task itself and not in any of its subroutines. That is why you cannot use a function like put_string() (lab 4) to output the string.

Compile your program and verify that there are no errors. Bring your code and evaluation board to your assigned lab period.

Procedure: Use a DB-9 serial cable to connect your 8051 board to a computer running a terminal emulator such as puTTY. Configure the terminal emulator for 8 data bits, 1 stop bit, no parity. Download and run your code. Verify the LED is mostly on but blinks off twice per second. Verify that messages alternate on the terminal emulator at a rate corresponding to the LED. Demonstrate this to your lab instructor.

Now, change the delay between the release of the first task and the second to 10 ms (1 tick). Recompile, download and run your program. Look at the output on the terminal emulator. What problem do you observe? Study your code and formulate an explanation for what you see. It may help to change the priority of one of the tasks. Record your explanation in your lab book.

Create a third semaphore and use it to control access to the serial port (i.e. add a call to OS_WaitBinSem() before accessing any of the serial port registers and a call to OSSignalBinSem() afterwards). Verify that with both tasks assigned the same priority the problem no longer occurs. Demonstrate your code to the lab instructor.

Affix all your source code to your lab book then write a summary or conclusion. Remember to sign or initial then date each page.