Embedded Control Systems, Winter 2009, Problem Set 7[1]

issued: Wednesday April 1, 2009 due: Monday April 13, 2009

To receive full credit for your answers to the following questions, please explain your reasoning carefully.

1. In the following code, the function `lSecondsSinceMidnight` returns the number of seconds since midnight. A hardware timer asserts an interrupt signal every second, which causes the microprocessor to run the interrupt routine `vUpdateTime` to update the static variables that keep track of time.

```
static int iSeconds, iMinutes, iHours
void interrupt vUpdateTime (void)
{
    ++iSeconds;
    if (iSeconds >= 60)
    {
        iSeconds = 0;
        ++iMinutes;
        if (iMinutes >= 60)
        {
            iMinutes = 0;
            ++iHours;
            if (iHours >= 24)
                iHours = 0;
        }
    }

          !! service the hardware

}

long lSecondsSinceMidnight (void)
{
    return( (((iHours * 60) + iMinutes) * 60) + iSeconds);
}
```

   (i) Is this code guaranteed to always return the correct answer? Explain.

   (ii) The ANSI C standard allows the compiled C code to read in the variables in any order, including the following: first `iHours`, then `iMinutes`, and lastly `iSeconds`. What is the maximum amount by which the returned value of time may be off? (Hint: Suppose that the time is 5:59:59.)

   (iii) Suggest a way to fix this code so that it returns the correct time.

[1] Revised March 30, 2009.

2. A *reentrant* function is one that may be called by more than one task, and will always work correctly even if the RTOS switches from one task to another in the middle of executing the function. The function `iFixValue` below uses the temporary variable `iTemp` to modify `iValue` which is shared by all the tasks that call it. Where would you need to take and release the semaphores to make the function below reentrant?

```
static int iValue;

int iFixValue (int iParm)
{
   int iTemp;

   iTemp = iValue;
   iTemp += iParm * 17;

   if (iTemp > 4922)
      iTemp = iParm;
   iValue = iTemp;

   iParm = iTemp + 179;
   if (iParm < 2000)
      return 1;
   else
      return 0;
}
```

3. Consider the problem of scheduling four tasks, with periods and execution times given as follows:

$$\begin{aligned} T_1: &\quad P_1 = 100, \quad e_1 = 20 \\ T_2: &\quad P_2 = 150, \quad e_2 = 30 \\ T_3: &\quad P_3 = 210, \quad e_3 = 80 \\ T_4: &\quad P_4 = 400, \quad e_4 = 100. \end{aligned}$$

   (a) Calculate the total utilization, U, for these four tasks. Do these tasks satisfy the sufficient condition

$$U < n(2^{1/n} - 1)$$

   for RMS schedulability?

   (b) If your answer to the preceding question is negative, does the answer change if you consider only the first three tasks? By eliminating tasks do you ever arrive at a combination that does satisfy the sufficient condition for schedulability?

   (c) Determine which tasks satisfy the necessary and sufficient condition for RMS schedulability by calculating the function

$$W_i(t) = \sum_{k=1}^{i-1} \lceil \frac{t}{P_k} \rceil e_k + e_i$$

   that determines the amount of time the CPU spends executing the tasks $T_1, \ldots, T_i$ in the interval $[0, t]$.

   (d) For those tasks that are schedulable, determine the times at which they complete by plotting which task is running at which time.

4. Consider the problem of scheduling three tasks with periods and execution times shown in Table 1. The deadline for each task is equal to its period.

Task	Period	Execution Time
1	$P_1 = 200$	$e_1 = 50$
2	$P_2 = 400$	$e_2 = 100$
3	$P_3 = 600$	$e_3 = 300$

Table 1: Three Periodic Tasks

   (a) Compute the total utilization for these three tasks. Do they meet the sufficient condition for Rate Monotonic (RM) schedulability? Explain.

   (b) Are Tasks 1 and 2 RM schedulable (calculate $W_2(t)$)?

   (c) Are Tasks 1, 2, and 3 RM schedulable?

   (d) Sketch the times at which the three tasks will be running under the rate monotonic scheduling protocol.