CPE355 - Real time embedded kernels - Spring'11
Prof. Nuno Alves (nalves@wne.edu), College of Engineering
Homework Assignment #13
Due date: 11am, Tuesday, March 13th 2012

WESTERN NEW ENGLAND UNIVERSITY | WNE

Question #1 (20 points)

The following two routines are called by Tasks A, B and C, but they don't work. How would you fix the problems?

```
static int iRecordCount;

void increment_records (int iCount)
{
   OSSemGet (SEMAPHORE_PLUS);
   iRecordCount += iCount;
}

void decrement_records (int iCount)
{
   iRecordCount -= iCount;
   OSSemGive (SEMAPHORE_MINUS);
}
```

Question #2 (20 points)

Where do you need to take and release the semaphores in the following code to make the function 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;
}
```

Question #3 (20 points)

For each of the following situations, discuss which of the three shared-data protection mechanisms seems most likely to be best and explain why.

(a.) Task M and Task N share an `int` array, and each often must update many elements in the array.

(b.) Task P shares a single `char` variable with one of the interrupt routines.

Question #4 (20 points)

Assume that the following code is the only code in the system that uses the variable `iSharedDeviceXData`. The routine `vGetDataFromDeviceX` is an interrupt routine. Now suppose that instead of disabling all interrupts in `vTaskZ`, as shown below, we disable only the device X interrupt, allowing all other interrupts. Will this still protect the `iSharedDeviceXData` variable? If not, why not? If so, what are the advantages (if any) and disadvantages (if any) of doing this compared to disabling all interrupts?

```
int iSharedDeviceXData;

void interrupt vGetDataFromDeviceX (void)
{
   iSharedDeviceXData = !! Get data from device X hardware
   !! reset hardware
}
void vTaskZ (void)   /* Low priority task */
{
   int iTemp;

   while (FOREVER)
   {
      .
      .
      .
      !!disable interrupts

       iTemp = iSharedDeviceXData;
       !!enable interrupts
       !!compute with iTemp
    }
 }
```

Question #5 (20 points)

Consider this statement: "In a nonpreemptive RTOS, tasks cannot 'interrupt' one another; therefore there are no data-sharing problems among tasks." Would you agree with this?