Jonathan W. Valvano First Name: _____ Last Name:_____
March 1, 2013, 10:00 to 10:50am

Quiz 1 is a closed book exam. You may have one 8.5 by 11 inch sheet of hand-written crib notes, but no books or electronic devices. You may put answers on the backs of the pages.

(10) Question 0. Please staple your crib sheet to your exam. Your crib sheet will be graded on content and correctness.

(10) Question 1. Assume a priority scheduler with blocking semaphores. Threads can have priority from 0 (highest) to 15 (lowest). The OS solves starvation using aging. The priority number of a thread which does not run is decremented once every t_1 seconds. Once it runs, the priority is restored to its regular level. The scheduler runs every t_2 seconds. There are at most n threads active at a time. Assume a particular thread has a regular priority of m, where $0 \leq m \leq 15$. Assuming the thread does not block, kill, or sleep, derive an equation to calculate the maximum time in between executions of this thread of priority m.

(5) Question 2. Give the two most important factors for effective debugging.

(20) Question 3. Select the best term that describes each definition.

A formal model that can be used to study data flow where nodes are connected using FIFO queues, such that we can guarantee that the FIFOs never become full.	
Software execution that cannot be divided or interrupted. Once started, the operation will run to its completion without interruption.	
The condition where once a thread begins to wait on a resource, there are a finite number of threads that will be allowed to proceed before this thread is allowed to proceed.	
A process where a governing body (e.g., FDA, FCC, DOD etc.) gives approval for the use of the device. It usually involves demonstrating the device meets or exceeds safety and performance criteria.	
A scheduling algorithm with round robin order but varying time slice. If a thread blocks on I/O, its time slice is reduced. If it runs to completion of a time slice, its time slice is increased.	
A scenario that occurs when two or more threads are all blocked each waiting for the other with no hope of recovery.	
An indirect function call added to a software system that allows the user to attach their programs to run at strategic times, created at run time and do not require recompiling the entire system.	
A characteristic when the presence of the collection of information itself does not affect the parameters being measured.	
A software technique to guarantee subfunctions within a module are executed in a proper sequence. For example, it forces the user to initialize I/O device before attempting to perform I/O.	
The percentage of resource utilization below which the RTOS can guarantee that all deadlines will be met.	

(10) Question 4. Consider this example showing a foreground and background thread. The assembly code generated by the compiler follows.

```
unsigned long Sec,Min;
void Foreground(void){
  printf("%02d:%02d/n", Sec, Min);
}
```

```
void Timer0A_Handler(void){
  TIMER0_ICR_R = TIMER_ICR_CAECINT;
  Sec = Sec + 1;
  if(Sec == 60){
    Sec = 0; Min++;
  }
}
```

```
0x00000128         Foreground
0x00000128 B510         PUSH   {r4,lr}
0x0000012A 48FE         LDR    r0,[pc,#1016]  ;address of Min
0x0000012C 6802         LDR    r2,[r0,#0x00]  ;value of Min
0x0000012E 48FE         LDR    r0,[pc,#1016]  ;address of Sec
0x00000130 6801         LDR    r1,[r0,#0x00]  ;value of Sec
0x00000132 A0FE         ADR    r0,{pc}+2      ;pointer to strg
0x00000134 F000FFD2     BL     printf
0x00000138 BD10         POP    {r4,pc}
0x0000013A         Timer0A_Handler
0x0000013A 2004         MOVS   r0,#0x04       ;TIMER_ICR_CAECINT
0x0000013C 49FD         LDR    r1,[pc,#1012]  ;address of TIMER0_ICR_R
0x0000013E 6248         STR    r0,[r1,#0x24]  ;Clear interrupt trigger, ack
0x00000140 48F9         LDR    r0,[pc,#996]   ;address of Sec
0x00000142 6800         LDR    r0,[r0,#0x00]  ;value of Sec
0x00000144 1C40         ADDS   r0,r0,#1       ;Sec+1
0x00000146 49F8         LDR    r1,[pc,#992]   ;address of Sec
0x00000148 6008         STR    r0,[r1,#0x00]  ;new value of sec
0x0000014A 4608         MOV    r0,r1
0x0000014C 6800         LDR    r0,[r0,#0x00]  ;value of Sec
0x0000014E 283C         CMP    r0,#0x3C       ;equal to 60?
0x00000150 D106         BNE    done
0x00000152 2000         MOVS   r0,#0x00
0x00000154 6008         STR    r0,[r1,#0x00]  ;Sec = 0
0x00000156 48F3         LDR    r0,[pc,#972]   ;address of Min
0x00000158 6800         LDR    r0,[r0,#0x00]  ;value of Min
0x0000015A 1C40         ADDS   r0,r0,#1       ;Min+1
0x0000015C 49F1         LDR    r1,[pc,#964]   ;address of Min
0x0000015E 6008         STR    r0,[r1,#0x00]  ;new value of Min
0x00000160 4770    done BX     lr
0x0000052C         strg DCB  0x25,0x30,0x32,0x64,0x3A,0x25,0x30,0x32,0x64,0x0A,0x00
```

Part a) There is a critical section. Explain the consequences of this critical section by giving a list of four or five **`printf`** outputs illustrating what mistake could occur. This prints **`Sec`**:**`Min`**.

Part b) Specify the exact beginning and end of the critical section by adding two arrows pointing into the assembly code.

(20) Question 5. Threads t1 t2 and t3 are foreground threads running with a round robin scheduler. In this system, each thread initializes SysTick before reading the counter. The goal of this problem is to define semaphores and add shared variables such that **NVIC_ST** registers are initialized exactly once. For example, t2 finishes execution of **SysTick_Init** first, in which case t1 and t3 will skip over its initialization steps. For example, t1 and t2 may never execute, in which case t3 executes **SysTick_Init** then reads **NVIC_ST_CURRENT_R**. over and over. It is important not to execute the initialization twice, and you must prevent a thread from reading the counter before it has been initialized. You cannot add/move any reads or writes to the **NVIC_ST** registers. You may make calls to **OS_Wait** and **OS_Signal** without showing the implementation of these two. You must use semaphores. Other than inside **OS_Wait**/**OS_Signal**, you are not allowed to disable interrupts. If you add semaphores or variables, give them good names and specify initial values.

```
// Add semaphores and variables here


void SysTick_Init(void){









  NVIC_ST_CTRL_R = 0;             // disable SysTick
  NVIC_ST_RELOAD_R = 0x00FFFFFF;  // maximum reload value
  NVIC_ST_CURRENT_R = 0;          // clears SysTick counter
  NVIC_ST_CTRL_R = 0x05;          // enable SysTick with core clock









}
```

```
void t1(void){
unsigned long n,t[256];
 n = 0;
 SysTick_Init();
 while(1){
   t[n]=NVIC_ST_CURRENT_R;
   n = (n+1)&0xFF;
   // other stuff
  }
}
```

```
void t2(void){
unsigned long n,t[256];
 n = 0;
 SysTick_Init();
 while(1){
   t[n]=NVIC_ST_CURRENT_R;
   n = (n+1)&0xFF;
   // other stuff
  }
}
```

```
void t3(void){
unsigned long n,t[256];
 n = 0;
 SysTick_Init();
 while(1){
   t[n]=NVIC_ST_CURRENT_R;
   n = (n+1)&0xFF;
   // other stuff
  }
}
```

(25) Question 6. You may assume someone else has written code that maintains a sorted list of TCBs containing active threads. This list is not circular; it is a simple linear list. There is an **`ActivePt`** that points to the front of this linked list, which is the highest priority active thread. Blocking semaphores are used. If a thread is sleeping, blocked, or killed someone else removes it from this active list. Furthermore, there cannot be two threads of the same priority. There is a **`RunPt`** that points to thread that is currently running. You will write the PendSV handler that is used to suspend the currently running thread, find the next thread to run, and launch the new thread. The launched thread may or may not be the same thread that was just suspended. The TCB structure IS REVERSED from the one in the book. The TCB structure is

```
struct tcb{
  struct tcb *next;  // linked-list pointer
  long *sp;          // pointer to stack, valid for threads not running
};
typedef struct tcb tcbType;
```

Write the assembly code for the PendSV handler that implements the thread switch for this priority scheduler. You do not have to modify the links in the linear linked list, simply suspend the running thread, choose the best thread to run, and launch the new thread.