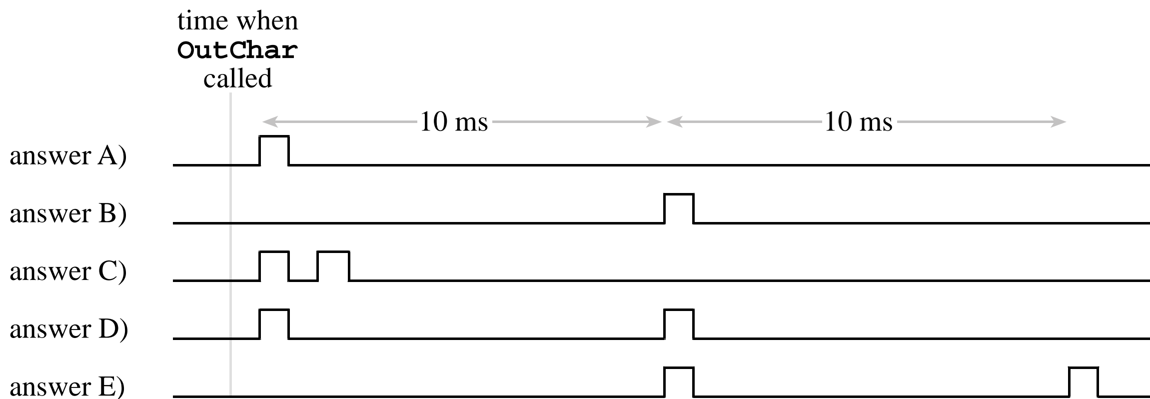
Jonathan W. Valvano First Name: _______________ Last Name:____________________
February 26, 2010, 10:00 to 10:50am

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, and please staple the crib sheet to your exam.

(6) Question 1. The following is copied from the serial port starter application. It has been edited to show only serial port output. The two lines in **bold** were added for debugging.

```
void USART1_IRQHandler (void) { volatile unsigned int IIR;
  struct buf_st *p;
  GPIOB->ODR |= 0x1000;    // bit 12 on LED
  IIR = USART1->SR;
  if (IIR & USART_FLAG_TXE) {
    USART1->SR &= ~USART_FLAG_TXE;         // clear interrupt
    p = &tbuf;
    if (p->in != p->out) {
      USART1->DR = (p->buf [p->out & (TBUF_SIZE-1)] & 0x1FF);
      p->out++;
      tx_restart = 0;
    }
    else {
      tx_restart = 1;
      USART1->CR1 &= ~USART_FLAG_TXE;  // disable TX interrupt
    }
  }
  GPIOB->ODR &= ~0x1000;   // bit 12 off LED
}
int OutChar (int c) {
  struct buf_st *p = &tbuf;   // If the buffer is full, return an error value
  if (SIO_TBUFLEN >= TBUF_SIZE)  return (-1);
  p->buf [p->in & (TBUF_SIZE - 1)] = c;   // Add data to the transmit buffer.
  p->in++;
  if (tx_restart) {           // If transmit interrupt is disabled, enable it
    tx_restart = 0;
    USART1->CR1 |= USART_FLAG_TXE; // enable TX interrupt
  }
  return (0);
}
```

Assume the baud rate is 1000 bits/sec, and PortB bit 12 is an output connected to the logic analyzer. Also assume the USART1 hardware is initially idle, and its FIFO is empty. Which of the following measurements do you expect to observe on PB12 after **OutChar** is called once?



(24) Question 2. Select the best term from the word bank that describes each definition.

Part a) A technique to periodically increase the priority of low-priority threads so that low priority threads occasionally get run. The increase is temporary.

Part b) The condition where low priority threads never get run.

Part c) The condition where thread 1 is waiting for a unique resource held by thread 2, and thread 2 is waiting for a unique resource held by thread 1.

Part d) The condition where a thread is not allowed to run because it needs something that is unavailable.

Part e) The condition where once a thread blocks, there are a finite number of threads that will be allowed to proceed before this thread is allowed to proceed.

Part f) An operation that once started will run to completion without interruption

Part g) An implementation using a FIFO or mailbox that separates data input from data processing.

Part h) A technique that could be used to prevent the user from executing I/O on a driver until after the user calls the appropriate initialization.

Part i) A scheduling algorithm that assigns priority linearly related to how often a thread needs to run. Threads needing to run more often have a higher priority.

Part j) An OS feature that allows the user to run user-defined software at specific places within the OS. These programs are extra for the user's convenience and not required by the OS itself.

Part k) An OS feature that allows you to use the OS in safety-critical applications.

Part l) 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.

word bank

active	hook
aging	maximum latency
atomic	nonreentrant
blocked	normalized mean response time
bounded buffer	path expression
bounded waiting	preemptive scheduler
breakdown utilization	producer-consumer
certification	rate monotonic
critical section	reentrant
deadlock	rendezvous
earliest slack time first	round robin scheduler
exponential queue	sleeping
fork	spin lock
killed	starvation

(10) Question 3. There is a sampling capacitor at the input of most ADC converters. The ADC is started by temporarily connecting the input voltage to this capacitor using a transistor switch. This transistor switch is controlled by a trigger signal. This trigger establishes when the ADC is started. In a real-time data acquisition it is important to control timing of this trigger. List three ways you could configure the ADC to trigger on the STM32F103 (our microcontroller)

(5) Question 4. There are many names for **Signal** and **Wait**. Sort the following terms into two groups according to whether the term means **Signal** or the term means **Wait**.

Pend
Post
verhogen
probeer te verlagen

Wait	Signal

(5) Question 5. What would happen in your RTOS if the background task the user attached using `OS_AddButtonTask` (tamper button) were to call the spinlock `OS_Wait`? Assume TIM1 (the one used for `OS_AddPeriodicThread`) runs at priority 0, Systick (used for the preemptive thread switch) runs at priority 1, tamper runs at priority 2, USART1 runs at priority 2, and PendSV runs at priority 15. Assume also the semaphore value is 0. Pick the best answer.

A) The stack would be corrupted because the thread switcher would switch out this background task, causing the system to crash.
B) The tamper task will spin because the semaphore is 0, causing the USART1 ISR to be locked out. However, the periodic thread and the foreground threads (everything except tamper and USART1) will continue to run.
C) The tamper task will spin forever, locking out all other components. The system stops running.
D) The tamper task will spin and no foreground threads will run. The periodic thread will continue to run. If the periodic thread calls `OS_Signal`, the tamper will become unstuck, and everything will resume executing.
E) The tamper task will spin and no foreground threads will run. Periodical and USART1 threads will continue to run. If the USART1 thread calls `OS_Signal`, the tamper will become unstuck, and everything will resume executing.

(25) Question 6. Consider a system that employs a preemptive real-time OS like Labs 2,3. There are multiple threads that wish to output to the serial port. Consider this example with two foreground threads (**`thread1`** **`thread2`**) and one background thread (**`isr`**) that all call **`Output`**. **`Busy`** is a global variable, initialized to 0.

<table>
<tr>
<td><pre>void thread1(void){
long data;
  init1();
  for(;;){
    data = calc1();
    Output(1,data);
  }
}</pre></td>
<td><pre>void thread2(void){
long data;
  init2();
  for(;;){
    data = calc2();
    Output(2,data);
  }
}</pre></td>
<td><pre>// TIM2 interrupt
void isr(void){
long data;
  data = calc3();
  Output(3,data);
  TIM2->SR &= ~1;
}</pre></td>
</tr>
</table>

The first parameter of **`Output`** is an id, and the second parameter is a number to output. This program has one or more critical sections. Add code to the following implementation to remove the critical section(s). You are allowed to disable/enable interrupts for short periods, but not during the **`printf`**. You should not introduce new critical sections. You CAN NOT allow threads to block or spin. If the serial port is busy, then the output is simply skipped. You may not change any thread code or the **`Output`** function prototype. Basically you will add software to this existing **`Output`** function, but not **`thread1`**, **`thread2`** or **`isr`**. Be careful not to crash in the interrupt service routine. If you do not remember the exact assembly code, you can answer in pseudo-code.

```
int Busy=0;
void Output (int id, long num){


  if(Busy == 0){



   Busy = 1;



   printf("Id = %u, num = %d\r\n",id,num);



   Busy = 0;

   }

}
```

(25) Question 7. Solve the following synchronization problem using semaphores. All threads are running in the foreground using a preemptive round robin scheduler. Thread 1 will execute first. Thread 1 will attempt to **`fun1`** once. Thread 2 will attempt to **`fun2`** once. Thread 3 will wait for either **`fun1`** or **`fun2`** to complete, then execute **`fun3`** over and over. It is possible for **`fun1`** and **`fun2`** to A) both complete eventually; B) just one completes and the other hangs up forever; or C) both do not ever complete. If neither **`fun1`** nor **`fun2`** completes, then **`fun3`** is never executed. You may assume the following three semaphore functions are available.

```
// ******** Init ************
// initialize semaphore
// input:  pointer to a semaphore
// output: none
void Init(Sema4Type *semaPt, long value);

// ******** Wait ************
// decrement semaphore and block if less than zero
// input:  pointer to a counting semaphore
// output: none
void Wait(Sema4Type *semaPt);

// ******** Signal ************
// increment semaphore, wakeup blocked thread if appropriate
// input:  pointer to a counting semaphore
// output: none
void Signal(Sema4Type *semaPt);
```

Part a) Create the semaphores needed using **`Sema4Type`**.

Part b) Add calls to the semaphore functions as needed to implement the synchronization

`void thread1(void){`
 	
`  fun1();`
 	
`  OS_Kill();`
`}`
	`void thread2(void){`
	
	`  fun2();`
	
	`  OS_Kill();`
	`}`
		`void thread3(void){`
		`  for(;;){`
		
		`    fun3();`
		
		`  }`
		`}`