Jonathan W. Valvano February 26, 2010, 10:00 to 10:50am

(6) Question 1. Refer back to Lab 1 preparation question 2e). The serial port has a transmit data register and a transmit shift register. The answer is C because the first interrupt occurs right away and begins to send the one character. As soon as the serial port puts the first character into the shift register, the data register is empty and a second interrupt occurs. This second interrupt notices the FIFO is empty and disarms the serial port transmitter.

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

Part a) **Aging** is 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) **Starvation** is where low priority threads never get run.

Part c) **Deadlock** is 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) **Blocked** is the condition where a thread is not allowed to run because it needs something that is unavailable.

Part e) **Bounded waiting** is 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 **atomic** operation, once started, will run to completion without interruption

Part g) **Producer-consumer** or **bounded buffer** problem uses a FIFO or mailbox that separates data input from data processing.

Part h) We could use **path expression** to prevent the user from executing I/O on a driver until after the user calls the appropriate initialization.

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

Part j) A **hook** is 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 needs **certification** to be used the OS in safety-critical applications.

Part l) An **exponential queue** is 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.

(10) Question 3.

1) Blinky used continuous mode. A new ADC is triggered immediately after the last conversion is complete.

2) You can set up the ADC in single conversion, software start. A new ADC is triggered by software. This is like the way we used the ADC on the 9S12 in EE319K and EE345L.

3) You can set up the ADC in single conversion, hardware start. A new ADC is triggered by one of seven hardware events. The most common hardware is a periodic timer capture. This way the ADC is started exactly f_s times per second without jitter.

(5) Question 4. There are many names for **Signal** and **Wait**.

Wait	Signal
Pend (used in uCOS-II)	Post (used in uCOS-II)
probeer te verlagen	*verhogen*

(5) Question 5. What would happen if the background task called the spinlock **`OS_Wait`**?

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.

(25) Question 6. foreground threads and one background thread all call **`Output`**.

```
int Busy=0;
void Output (int id, long num){ long sr;  // a place to save I-bit
; assembly to read PRIMASK and save it in sr
; assembly to disable interrupts
  if(Busy == 0){
   Busy = 1;
; assembly to restore PRIMASK (I bit) from value saved in sr
   printf("Id = %u, num = %d\r\n",id,num);
   Busy = 0;
   return;
   }
; assembly to restore PRIMASK (I bit) from value saved in sr
}
```

(25) Question 7. Solve the following synchronization problem using semaphores.

Part a) You can create semaphores by defining them as variables with **`Sema4Type`**.

```
Sema4Type Ready;  // becomes true if either fun1 or fun2 complete
```

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

`void thread1(void){`
`  Init(&Ready,0)`
`  fun1();`
`  Signal(&Ready);`
`  OS_Kill();`
`}`
	`void thread2(void){`
	`  fun2();`
	`  Signal(&Ready);`
	`  OS_Kill();`
	`}`
		`void thread3(void){`
		`  for(;;){`
		`    Wait(&Ready);`
		`    fun3();`
		`    Signal(&Ready);`
		`  }`
		`}`