EE345M/EE380L Final Exam Spring 2010 Solution

Jonathan W. Valvano
May 17, 2010, 9am-12noon

(10) Question 1. Assume the Arm Cortex M3 is running with interrupts enabled using the PSP.
Part a) R0, R1, R2, R3, R12, PC, PSR, and LR (R14)
Part b) The Arm uses the main stack pointer (MSP) during the execution of the ISR.
Part c) R14 (LR) contains 0xFFFFFFF9 while the computer executes the ISR, 0xFFFFFFF9 means the computer is executing an ISR.
Part d) Yes, a higher priority ISR can interrupt a lower priority ISR. However, ISRs of the same level will not interrupt each other.
Part e) The STM32 employs software interrupt acknowledgement.

```
  TIM1->SR &= ~(1<<0); // clear UIF flag by writing a zero to it
  now = TIM3->CCR2;    // clear input capture CC2IF by reading latch
```

(15) Question 2. Four design choices one must make when implementing a spectrum analyzer.
1) ADC range. This defines the smallest and largest voltage that can be measured.
2) ADC precision. Range and precision define the voltage resolution of the system. Resolution is Range/Precision; where precision is given in alternatives
3) ADC sampling rate, f_s. The sampling rate defines the range of frequencies that can be measured: 0 to ½ f_s.
4) The buffer size n. The buffer size and the sampling rate together define the frequency resolution of the spectrum analyzer, $\Delta f = f_s/n$.

Other choices one could have made are FFT windowing, data format (fixed point, floating point), and zero padding the data.

(5) Question 3. The Discrete Fourier Transform (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn} \quad \text{where} \quad W_N = e^{-j2\pi/N} \quad k = 0,1,2,\ldots,N-1$$

(5) Question 4. Consider a PID control system for a linear system.
(2) Part a) You should run the controller 10 to 20 times faster than the time constant of the motor.
(2) Part b) Noise will directly affects controller accuracy, and may render D-term useless.
(1) Part c) Any time delay in a controller can cause instability (i.e., poles in right-half plane).

(5) Question 5. This software is not critical section because the read modify write to Port B is atomic. The software writes to BSRR, and the hardware sets the bit (you can also clear bits atomically).

(10) Question 6. In 16 words or less give a definition of the following terms
Part a) Internal fragmentation is wasted space inside block. E.g., allocating whole block when only part is needed.
Part b) Back EMF occurs in an inductor when current is shut off, V = L dI/dt.
Part c) Paging is address translation hardware mapping a logic address into a physical address.
Part d) Aging is a temporary increase in priority as a solution to starvation.
Part e) Dual address means two memory cycles: first read data from source then write data to destination.
Part f) Stuff bits are extra bits in USB and CAN preventing long sequences of zeros or ones.
Part g) Content addressable memory has input of data, and output is the address.
Part h) Input membership set is a fuzzy logic variable, 255 means true and 0 means false.
Part i) Hook is a indirect function call allowing dynamic insertion of code.
Part j) Bounded waiting is once a thread begins waiting, a finite number of threads will go first.

(30) Question 7. Let **Local** be the computer holding the counter. Let Remote be another computer, not holding the counter. Each **Remote** is given a unique Id, which is a power of 4, e.g., 4, 8, 12,… **`LocalReceiveCanMessages`** is a foreground thread that needs to be added to the scheduler by calling **OS_AddThread**. On the **Local** computer, signal and wait are standard spin-lock counting semaphores.

```
long Counter;  // official copy of semaphore
```

```
// ******** OS_InitNetSem ************
// initialize network semaphore
// inputs:  initial Counter value
// output: none
void OS_InitNetSem(long value){
  Counter = value;
}

// no CAN data is needed for this solution
// CAN Id = 0 means signal from another computer
//            sent by Remote, received by Local
// CAN Id 4,8,12,16... means request for wait
//            sent by Remote (Id), received by Local
// CAN Id+1 means wait is successful, Counter decremented
//            sent by Local, received by Remote(Id)
// CAN Id+2 means wait is unsuccessful, Counter not decremented
//            sent by Local, received by Remote(Id)
void LocalReceiveCanMessages(void){ OSCRITICAL_LOCALS()
unsigned short id;
  while(1){
    if(CAN_ReceiveMessage(&id)) OS_Suspend(); // cooperative
    else{
      OSCRITICAL_ENTER()     // atomic
      if(id==0) Count++;
      else if((id&0x03)==0){ // request for wait
        if(Count>0){
          Count--;           // success
          id = id+1;
        }else id = id+2;     // failure
        CAN_SendMessage(id);
      }
      OSCRITICAL_EXIT()
    }
  }
}

// ******** OS_LocalWait ************
// if Counter>0, decrement; if Counter is zero, spin
// input:  none
// output: none
void OS_LocalWait(void){
  OS_DisableInterrupts();     // Test and set is atomic
  while(Counter <= 0){        // disabled
    OS_EnableInterrupts();
    OS_DisableInterrupts();
  }
  Counter--;       // disabled
  OS_EnableInterrupts();   // disabled
}

// ******** OS_LocalSignal ************
// increment Counter
// input:  none
// output: none
void OS_LocalSignal(void){ OSCRITICAL_LOCALS()
  OSCRITICAL_ENTER()          // start critical section
  Counter++;                 // this is atomic
  OSCRITICAL_EXIT()          // end critical section
}
```

These two functions are called on a computer not containing the official counter.

```
#define ME 4    // 4 8 12 16 (different value for each Remote)
// ******** OS_NetWait ************
// if Counter>0, decrement; if Counter is zero, spin
// input:  none
// output: none
void OS_NetWait(void){ unsigned short id;
  CAN_SendMessage(ME);    // request for wait
  while(1){
    if(CAN_ReceiveMessage(&id)) OS_Suspend(); // cooperative
    else{
      if(id==ME+1) return;    // success
      if(id==ME+2)
        CAN_SendMessage(ME);  // failure, request for wait again
    }
  }
}
// ******** OS_NetSignal ************
// increment Counter
// input:  none
// output: none
void OS_NetSignal(void){
 CAN_SendMessage(0);  // id of 0 means signal
}
```

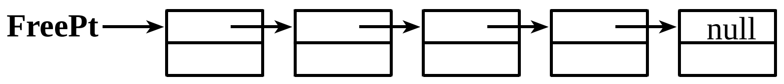
(20) Question 8. Implement a memory manager for fixed sized blocks. Let the block size be

```
#define SIZE 100   // size in bytes
```

Let the number of blocks be

```
#define NUM 10     // number of blocks
#define NULL 0     // empty pointer
char *FreePt;      // points to first free block
char Heap[SIZE*NUM];
```

Initialization must be performed before the heap can be used. **`Heap_Init`** partitions the heap into blocks and links them together. **`FreePt`** points to a linear linked list of free blocks. Initially these free blocks are contiguous and in order, but as the manager is used the positions and order of the free blocks can vary. It will be the pointers that will thread the free blocks together.



```
void Heap_Init(void){
char *pt;
  FreePt = &Heap[0];
  for(pt=&Heap[0]; pt!=&Heap[SIZE*(NUM-1)]; pt=pt+SIZE){
    *(long*)pt =(long)(pt+SIZE);
  }
  *(long*)pt = NULL;
}
```

To allocate a block to manager just removes one block from the free list. The **`Heap_Allocate`** function will fail and return a null pointer when the heap becomes empty. The **`Heap_Release`** returns a block to the free list. This system does not check to verify a released block actually was previously allocated.

```
void *Heap_Allocate(void){
char *pt;
  pt = FreePt;
  if (pt != NULL){
    FreePt = (char*) *(char**)pt;
  }
  return(pt);
}
```

```
void Heap_Release(void *pt){
char *oldFreePt;
  oldFreePt = FreePt;
  FreePt = (char*)pt;
  *(long*)pt = (long)oldFreePt;
}
```