CPE355 - Real time embedded kernels - Spring'11

Prof. Nuno Alves (nalves@wne.edu), College of Engineering

Midterm Exam #2 - Solutions

Date: 11am, Thursday, April 26th 2012

Each question is worth 5 points. Partially correct questions are worth 2 points.

Embedded C

1) What is the difference between = and == ?

```
 The symbol "=" is the assignment operator whereas "==" is the comparison
operator.
```

2) Using the following code skeleton, complete the function that sorts an array of 5 elements by reference.

```
 void sort_array(int array[], int nelements)
 {

 }
```

```
void sort(int array[], int nelements)
{
        int i,j=0;
        int temp;

        for (i=0;i<nelements;i++)
        for (j=0;j<nelements;j++)
        if (array[i]<array[j])
        {
                temp=array[i];
                array[i]=array[j];
                array[j]=temp;
        }

}
```

3. Consider the structure below. How can you assign the value 10 to the member i of the structure xxx using *z?

```
#include<stdio.h>

struct myStructure
{
   int i;
   char *p;
};

int main()
{
   struct myStructure xxx;
   struct myStructure *z=&xxx;

   //... something goes here ...
}
```

```
*z.i = 10; or z->i=10;
```

4. How do you de-reference the void pointer (`pVoid`) to extract its associated value? Print its associated value.

```
#include<stdio.h>

int main()
{
   float floatValue=55.5;

   void *pVoid; //declaring a void pointer

   pVoid = &floatValue; // pVoid points to the address of intValue

   //... one line is missing here ...


   //... next line prints the floatValue through the
   //de-referenced void pointer ...


}
```

```
   float *pFloat = (float*)(pVoid);
   printf("%f\n",*pFloat);
```

5. What is a function pointer? Write an example.

```
It is a pointer that points to a function.
```

```
#include <stdio.h>

float Plus    (float a, float b) { return a+b; }
float Minus   (float a, float b) { return a-b; }

void fPTR_example(float a, float b, float (*pt2Func)(float, float))
{
   printf("%f\n",pt2Func(a, b));
}

int main()
{
   fPTR_example(2, 5, &Minus);
   fPTR_example(2, 5, &Plus);
}
```

6. Write some code that will place the integer value 0X0034 into the absolute memory address 0xA3B5.

```
int *a;
a= (int *) 0xA3B5;
*a=0X0034;
```

7. Write a SUM macro which takes two arguments and returns the sum of the two.

```
#define SUM(A,B) (A+B)
```

8. What is a register variable and why do we need them?

```
Register variables are a special case of automatic variables. Automatic
variables are allocated storage in the memory of the computer; however,
for most computers, accessing data in memory is considerably slower than
processing in the CPU. By specifying a register variable, the programmer
suggests the compiler that particular automatic variables should be
allocated to CPU registers, if possible.
```

9. Write an example in which you modify the contents of a const int variable.

```
const int abc=10;
int *a=&abc;
*a=11;
```

10. Using the following structure, write a function that will count the number of elements of a linked list.

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
 int Data;
 struct Node *Next;
}*Head;

// You must complete this function that counts number of elements
int length()
{









  return(count);
}

int main(int argc, char *argv[])
{
 int i=0;

 //Set HEAD as NULL
 Head=NULL;
 addBeg(num); //this function has been declared somewhere else
 printf("Number of elements=%d\n",length());
};
```

```
// Counting number of elements in the List
int length()
{
  struct Node *cur_ptr;
  int count=0;

  cur_ptr=Head;

  while(cur_ptr != NULL)
  {
     cur_ptr=cur_ptr->Next;
     count++;
  }
  return(count);
}
```

RTOS

11. What are some of the common features offered in a commercial RTOS?

```
 Features such as intertask communication, timer services, memory
management and events.
```

12. Each task has its own private context. What does this mean?

It means each task includes their private register values, a program counter, and a stack. This data is not shared with other tasks in the system.

13. In a RTOS, what is the state of a task that is waiting for an event to occur?

Blocked.

14. Interrupt latency, footprint and context switching time are perhaps the most important factors considered when a RTOS is being selected. Could you briefly describe them?

- Interrupt latency: the time it takes to process a given interrupt
- Footprint: the size of the executable code generated after compilation
- Context switching time: the time is takes for tasks to switch between different states.

15. What happens if the programmer disables all maskable interrupts and forgets to re-enable it?

The processor will execute no more interrupt routines until they are re-enabled. When interrupts are reenabled, they will sequentially be processed by priority order.

16. What is an atomic operation?

A single-line operation that cannot be interrupted.

17. What is priority inversion?

It's the situation whenever a higher priority task has to unnecessarily wait for a lower priority task to finish. This happens when the higher priority task has to wait for a shared resource (e.g. a semaphore) to be released by a lower priority task that is currently running.

18. What is a non-preemptive scheduling (a.k.a. cooperative multi-tasking) algorithm?

In this type of scheduling once a task is in the running state, it doesn't leave that state until it is done, or until it has no further resources to continue computation.

19. What is a reentrant function?

Reentrant functions are functions that can be called by more than one task and that will always work correctly, even if the RTOS switches from one task to another in the middle of executing the function.

20. Why RTOS developers tend to avoid the standard calloc and malloc functions?

Two standard memory allocation C functions are malloc and free: real-time systems often avoid these two functions because they are slow their execution times are unpredictable. Most RTOSs offer fast and predictable functions that allocate and deallocate memory. Malloc and calloc also contribute to memory fragmentation issues.