EECS 498 homework 2: *Due 10/24 by 3pm*

The following is to be done individually and your answers are to be typed where viable (diagrams and the like can be drawn freehand). The assignment must be stapled.
There are 4 parts for 55 points. The first three parts should really take no longer than two hours total. Part 4 may take an hour or two by itself.

Part 1: Definitions [10 points]

Define the following terms: **[1 point each]**

a. Mil (PCB measurement)
b. Thou (PCB measurement)
c. Neck down (PCB)
d. Rat's nest (PCB)
e. Real-time system
f. Hard deadline
g. Soft deadline
h. Firm deadline
i. Wear leveling (in Flash memory)
j. Priority inversion

Part 2: Short answer [12 points]

Answer the following questions in something between a sentence and a short paragraph:
[3 points each]

1. What are the advantages and disadvantages of using GPL code in your embedded application?
2. What are the advantages of using busybox on an embedded Linux application?
3. How does priority inheritance address priority inversion?
4. What is a "deferred interrupt" and why are they useful?

Part 3: Longer answer [25 points]

1. In this class we have discussed three basic schemes for controlling software on an embedded system (barebones, RTOS, desktop OS). Briefly (say less than one page) compare and contrast these options and then generate a scenario for each scheme where it is clearly preferred over the other two. **[8 points]**
2. What are the pros and cons of having a MMU and virtual memory on an embedded system? In particular address why MMUs are rare on low-end systems but sometimes highly desirable on larger systems. **[6 points]**
3. Discuss the pros and cons of EDF and RMS. **[5 points]**
4. Answer the question on the next page. **[6 points]**

Consider the following code for a Linux module.

```
module_init(memory_init);
module_exit(memory_exit);

struct file_operations fops = {
      .read = memory_read,
      .write = memory_write,
      .open = memory_open,
      .release = memory_release
};
int memory_major = 60;
char *memory_buffer;

int memory_open (struct inode *inode,       struct file *filp) {
      printk("<1> Minor: %d\n",
                MINOR(inode->i_rdev));
     return 0;
}

int memory_init(void) {
     int result;
     result = register_chrdev(memory_major, "memory",
&memory_fops);
     if (result < 0) {
         printk("<1>memory: cannot obtain major number %d\n",
                  memory_major);
         return result;
}
int memory_release (struct inode *inode,          struct file
*filp) {
return 0;
}
```

Once the Linux device is correctly installed, the memory_open() function will be called each time the device is newly used (say by piping information to the device). How does the Linux system figure out what function to call when first opening the device? Be sure to include mention of all functions/data structures and what their role is.

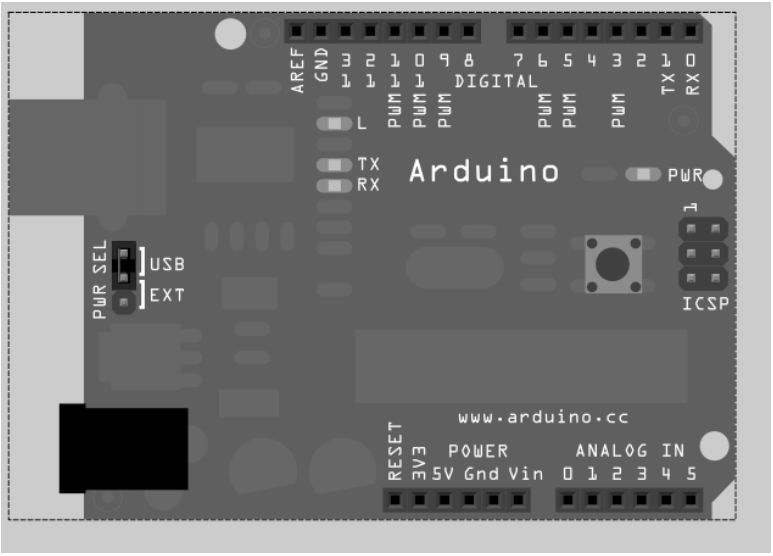
Part 4: Working with parts [18 points]

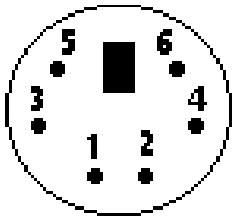
You are working on prototyping a device that is supposed to act like a PS2 keyboard. That is, it will be using the PS2 interface to send messages to a PC as if it were a keyboard. You've learned that there is a startup sequence you need to respond to in order to get the PC to know you are connected. As a hack (for the moment) you are hooking up a standard keyboard during the boot time, disconnecting it, and then attaching your device.

All you want to do is send an ASCII "A" (hex 41) to the PC *about* once a second. You are to design the hardware and write and an Arduino sketch, which accomplishes this task. You will need to read about the PS2 protocol on-line.

HINT: upon doing a web search, you have learned you can make an Arudino I/O pin be "HiZ" by making the pin be an input.

1. Draw the needed connections between the Arduino and the male 5-pin connector. Add components as needed (you will lose points for adding unneeded components). **[3 points]**





1.

2. Write the Arduino sketch that will accomplish the desired task. You are to bit-bang this out and you may assume "digitalWrite" and "pinMode" take well less than a microsecond each. Don't worry overly much about exact syntax.
   [15 points]