CS414 Assignment 2
10 points per answer for 1a-d, then 60 points for question 2 (100 max). Due Wednesday February 8

Short-answer questions (just a sentence or two each, please).

1. Suppose that you are writing an operating system for a standard PC and wanted to build in some logic for determining whether or not the OS was actually running in a virtual machine monitor as opposed to running on the "raw" hardware. Here are some ideas for how you could do that. Indicate for each one whether or not it would work, and give a short reason. (If the idea includes a completely wrong assertion, explain what is wrong about it).
    a. The O/S could include some sort of compute-intensive program. By testing on the same platform, the O/S should have a pretty good idea of how long this takes to run on a raw CPU and can compare with the timing it actually sees. If the O/S is running in a virtual machine, this will slow things down so much so that it will be obvious.
    b. Same idea as in (1) but with a focus on the speed at which system calls are executed – e.g. the O/S would measure some simple system call, like "gettimeofday()", doing perhaps ten million of them, and look for evidence that a VMM might be intercepting the traps and then reissuing them.
    c. The O/S can just write something to the disk and read it back in. If the disk is a virtualized, these operations will fail.
    d. In a VMM, the O/S won't get interrupts, so it will be obvious.

2. Suppose that the operating system has a disk driver coded in C that maintains a list of pending I/O operations. Nodes look like this:

```
struct   disk_io_req {
         struct disk_io_req  *next;  /* pointer to the next one, or null */
         struct disk_io_req *last;   /* pointer to the last one, or null */
         int      disk_io_op;        /* 0 for READ, 1 for WRITE */
         void   *buffer_addr;        /* Start address for transfer */
         int      buffer_len;        /* Length in bytes */
} *disk_io_list;
```

   As you can see, the disk_io_list pointer is either NULL (no work to do), or points to a circular list with each io request object linked to the next one in the list and to the previous one in the list. If there is just a single node, it points to itself. Here's the code used to add a new request to the list and to remove one; these are called within the operating system as the need arises:

```
void add_disk_io_req(struct disk_io_req *dr)
{
```

```
        if(disk_io_list == NULL) /* If the list was empty… */
                disk_io_list = dr->next = dr->last = dr;
        else /* … if not, put the new node at the very end of the list */
        {
                dr->next = disk_io_list;
                dr->last = disk_io_list->last;
                disk_io_list->last->next = dr;
                disk_io_list->last = dr;
        }
}

struct disk_io_req *get_disk_io_req()
{
        struct disk_io_req *dr = disk_io_list;
        if(dr == NULL) return(dr);  /* empty */
        if(dr->next == dr)   /* list has just a single item on it */
        {
                disk_io_list = NULL;
                return(dr);
        }
        /* general case: remove the first item and return it */
        disk_io_list = dr->next;
        dr->next->last = dr->last;
        dr->last->next = dr->next;
        return(dr);
}
```

Notice that the operating system calls add_disk_io_req() and get_disk_io_req() from time to time. Would it be safe to also call get_disk_io_req() from an interrupt handler? We're thinking about a situation where the disk has a list of things to do, and one of them finishes, and we want to start doing the next. If so, explain why. If not, explain what might go wrong. In your answer assume that no changes are made to this code: what you see above is precisely what would get executed.