```
CS 202, Spring 2015
Handout 8 (Class 10)

1. Example use of I/O instructions: boot loader

    Below is a WeensyOS boot loader (for WeensyOS Schedos, which is part
    of lab 4).

    It may be helpful to understand the overall picture (given in the
    comments; see also our class notes l03, Section 2(b).)

    The main point for today's class is that this code demonstrates I/O,
    specifically with the disk: the bootloader reads in the kernel from
    the disk.

    See the functions waitdisk() and readsect(). Compare to Figures 36.4
    and 36.5 in OSTEP.

/* boot.c */

#include "x86.h"
#include "elf.h"

/*********************************************************************
 * This a dirt simple boot loader, whose sole job is to boot
 * an ELF kernel image from the first IDE hard disk.
 *
 * DISK LAYOUT
 *  * This program (bootstart.S and boot.c) is the bootloader.
 *    It should be stored in the disk's sector 0 (the first sector).
 *
 *  * Sectors 1 through 32 hold the image for the schedos kernel and the
 *    schedos application images (as binaries).
 *
 *  * The kernel image must be in ELF executable format.
 *
 * BOOT UP STEPS
 *  * When the CPU boots it loads the BIOS into memory and executes it.
 *
 *  * The BIOS intializes devices, sets up the interrupt routines, and
 *    reads the first sector of the boot device (e.g., hard-drive)
 *    into memory and jumps to it.
 *
 *  * Assuming this boot loader is stored in the first sector of the
 *    hard-drive, this code takes over.
 *
 *  * Control starts in bootstart.S, which sets up protected mode,
 *    and a stack so C code then run, then calls bootmain().
 *
 *  * bootmain() in this file takes over, reads in the kernel image,
 *    and jumps to it.
 *
 *********************************************************************/

#define SECTORSIZE      512
#define PAGESIZE        4096
#define ELFHDR          ((struct Elf *) 0x10000) // scratch space

void readsect(void *addr, uint32_t sect);
void readseg(uint32_t va, uint32_t filesz, uint32_t memsz, uint32_t sect);

void
bootmain(void)
{
    struct Proghdr *ph, *eph;
    uint32_t *stackptr;

    // read 1st page off disk
    readseg((uint32_t) ELFHDR, PAGESIZE, PAGESIZE, 1);

    // is this a valid ELF?
    if (ELFHDR->e_magic != ELF_MAGIC)
            return;
```



```

    // load each program segment (ignores ph flags)
    ph = (struct Proghdr*) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
    eph = ph + ELFHDR->e_phnum;
    for (; ph < eph; ph++)
            readseg(ph->p_va, ph->p_filesz, ph->p_memsz,
                    1 + ph->p_offset / SECTORSIZE);

    // jump to the kernel, clearing %eax
    __asm __volatile("movl %0, %%esp; ret" : : "r" (&ELFHDR->e_entry), "a" (0));
}

// Read 'filesz' bytes at 'offset' from kernel into virtual address 'va',
// then clear the memory from 'va+filesz' up to 'va+memsz' (set it to 0).
void
readseg(uint32_t va, uint32_t filesz, uint32_t memsz, uint32_t sect)
{
        uint32_t end_va;

        end_va = va + filesz;
        memsz += va;

        // round down to sector boundary
        va &= ~(SECTORSIZE - 1);

        // read sectors
        while (va < end_va) {
                readsect((uint8_t*) va, sect);
                va += SECTORSIZE;
                sect++;
        }

        // clear bss segment
        while (end_va < memsz)
                *((uint8_t*) end_va++) = 0;
}

void
waitdisk(void)
{
        // wait for disk reaady
        while ((inb(0x1F7) & 0xC0) != 0x40)
                /* do nothing */;
}

void
readsect(void *dst, uint32_t sect)
{
        // wait for disk to be ready
        waitdisk();

        outb(0x1F2, 1);         // count = 1
        outb(0x1F3, sect);
        outb(0x1F4, sect >> 8);
        outb(0x1F5, sect >> 16);
        outb(0x1F6, (sect >> 24) | 0xE0);
        outb(0x1F7, 0x20);      // cmd 0x20 - read sectors

        // wait for disk to be ready
        waitdisk();

        // read a sector
        insl(0x1F0, dst, SECTORSIZE/4);
}

```

```
2. Two more examples of I/O instructions

    (a) Reading keyboard input

    The code below is an excerpt from WeensyOS's x86.c.

    This reads a character typed at the keyboard (which shows up on the
    "keyboard data port" (KBDATAP)), and converts it to a digit. This
    code is not called in lab4; it was called in lab1. (Use grep to
    convince yourself of this!)

    /* Excerpt from WeensyOS x86.c. Comments from kbd.h in xv6 */

    #define KBSTATP 0x64  // keyboard controller status port (I)
    #define KBS_DIB 0x01  // keyboard data in buffer
    #define KBDATAP 0x60  // keyboard data port (I)

    int
    console_read_digit(void)
    {
            uint8_t data;

            if ((inb(KBSTATP) & KBS_DIB) == 0)
                    return -1;

            data = inb(KBDATAP);
            if (data >= 0x02 && data <= 0x0A)
                    return data - 0x02 + 1;
            else if (data == 0x0B)
                    return 0;
            else if (data >= 0x47 && data <= 0x49)
                    return data - 0x47 + 7;
            else if (data >= 0x4B && data <= 0x4D)
                    return data - 0x4B + 4;
            else if (data >= 0x4F && data <= 0x51)
                    return data - 0x4F + 1;
            else if (data == 0x53)
                    return 0;
            else
                    return -1;
    }

    (b) Setting the cursor position

    The code below is also excerpted from WeensyOS's x86.c.
    It clears the console (see next items on the handout) and then uses
    I/O instructions to set a blinking cursor in the upper left of the
    screen.

    /*
     * console_clear
     *
     *   Clear the console by writing spaces to it, and move the cursor
     *   to the upper left (row 0, column 0).
     */

    void
    console_clear(void)
    {
            int i;

            /* what's this?? (see next items on handout) */
            cursorpos = (uint16_t *) 0xB8000;

            for (i = 0; i < 80 * 25; i++)
                    cursorpos[i] = ' ' | 0x0700;

            outb(0x3D4, 14);       // Command 14 = upper byte of position
            outb(0x3D5, 0 / 256);  // row 0
            outb(0x3D4, 15);       // Command 15 = lower byte of position
            outb(0x3D5, 0 % 256);  // column 0
    }
```



```
3. Memory-mapped I/O

    a. Here is a 32-bit PC's physical memory map:

        +------------------+  <- 0xFFFFFFFF (4GB)
        |      32-bit      |
        |  memory mapped   |
        |     devices      |
        |                  |
        /\/\/\/\/\/\/\/\/\/\

        /\/\/\/\/\/\/\/\/\/\
        |                  |
        |      Unused      |
        |                  |
        +------------------+  <- depends on amount of RAM
        |                  |
        |                  |
        | Extended Memory  |
        |                  |
        |                  |
        +------------------+  <- 0x00100000 (1MB)
        |     BIOS ROM     |
        +------------------+  <- 0x000F0000 (960KB)
        |  16-bit devices, |
        |  expansion ROMs  |
        +------------------+  <- 0x000C0000 (768KB)
        |   VGA Display    |
        +------------------+  <- 0x000A0000 (640KB)
        |                  |
        |    Low Memory    |
        |                  |
        +------------------+  <- 0x00000000

    [Credit to Frans Kaashoek, Robert Morris, and Nickolai Zeldovich for
    this picture]


    b. Loads and stores to the device memory "go to hardware".

    An example is in the console printing code from WeensyOS. Here are
    excerpts from lib.h and lib.c:

    /* Compare the addresses below to the map above. */
    #define CONSOLE_BEGIN   ((uint16_t *) 0x000B8000)
    #define CONSOLE_END     (CONSOLE_BEGIN + 80 * 25)


    /*
     * prints a character to the console at the specified
     * cursor position in the specified color.
     * Question: what is going on in the check
     *       if (c == '\n')
     * ?
     * Hint: '\n' is "C" for "newline" (the user pressed enter).
     */
    static uint16_t *
    console_putc(uint16_t *cursor, unsigned char c, int color)
    {
            if (cursor >= CONSOLE_END)
                    cursor = CONSOLE_BEGIN;
            if (c == '\n') {
                    int pos = (cursor - CONSOLE_BEGIN) % 80;
                    /* what does this do? */
                    for (; pos != 80; pos++)
                            *cursor++ = ' ' | color;
            } else
                    *cursor++ = c | color;
            return cursor;
    }
```