HOMEWORK 3

A functional bootloader

Issued	2/11/14
Due Date	2/20/14 **before class**

NOTE: As a reminder, homework is handed in when it is committed to your personal subversion repository before the deadline. Changes made after the deadline will be ignored.

Goal

Since we decided to build a fully functional Operating System, we need a working bootloader for our system. The goal of this homework is to implement a working bootloader, which will load our kernel from floppy and hand over control to the kernel.

Evaluation

The homework will be evaluated using the following factors:

- Correctness of assembler syntax -- The code should compile without error using **nasm**.
- Documentation of the code -- The code is clearly documented, explaining what the intent is and how it is accomplished.
- Correctness of execution -- The code performs the functions requested.
- Creativity -- Your solution is different. Make sure to **document** your code or we might miss your special trick!
- Optional tasks -- You implemented some or all of the optional requests, or went beyond what was requested in some other way.

You are required to check in your assembly code in subversion. The state of the repository at the beginning of class on the due date will be used for evaluation.

What to hand in

As for all the other homework assignments, you will need to commit your homework to your assigned subversion repository, under the correct directory (`homework/homework3`).

Please commit the following files:

- The assembly source file (or files) (`.asm`)
- The floppy disk image (`.vfd`)
- Your `mkboot.c` file
- Makefile used to build your floppy disk image
- A text file `DESCRIPTION` containing a description of what your code does, how you achieved it, clearly listing which tasks (see above) you completed. This should include (but not be limited to) a description of where the `mkboot` program has put the kernel on the floppy and a description of how your bootsector determines which sectors contain the kernel to be loaded.
- A document showing the results (references and summary) of your information search.

Task 1: Bootloader

The end result of this task should be a floppy image containing a function boot loader and kernel. The kernel will be provided (link will follow). For now, you can just use another file with a size between 200K and 300K.

This task has two components:

- Our boot sector and boot loader
- A tool to create the floppy image.

The latter can be written in C (or C++) or any other language you would prefer. It should take your floppy image, and put the specified kernel on it, updating the boot sector so that it can load the kernel at boot time.

NOTE: You can assume we won't be working with disks larger than 4294967296 sectors (2 TB), so you can assume the sector number fits in a 32 bit number.

Bootloader

You can start from your own bootsector or from the example solution in the shared repository. There is no need to keep the code regarding time or name.

Since we don't have a file system yet, the bootloader will use a list of sector numbers (placed on the floppy by the mkboot tool -- see other task). It will load each of these sectors, starting at offset `0x10000`. **Your code should verify that there is enough memory to do so**. While loading the kernel, a `.` (dot) should be output for each sector read.

Some hints:

- Think about how the bootloader will figure out which sectors to load from the floppy. It doesn't understand the filesystem of the floppy (there is none in this case). So, it will rely on the `mkboot` (below) tool to provide that information.
- The bootloader can only assume it can access sector 0; All other sectors belong to the file system. So, the `mkboot` tool will have to put information about where to find the kernel (on the floppy) in the bootsector.
- The kernel is not guaranteed to be stored contiguous on the floppy; You can assume that a sector either contains metadata (helps organizing the file system) or data (kernel) but no mixture of both.
- The kernel can be up to 1MB in size. A listing of all of the sector numbers holding kernel data will not fit in the bootsector. You may assume that one or more data sectors will be used to store this list.
- You can use `INT13h` subfunction `02h` to access the floppy. (Read up on LBA and C/H/S and make sure you understand why LBA was introduced)
- Remember how segments work in real mode. If the kernel you're loading is larger than 64K, you will have to change the segment register as you are copying.

mkboot

We need to create a tool that, given a floppy image and a kernel, can put the kernel on the floppy and update the boot sector (to tell the bootloader where to look for the kernel).

This tool runs on your development system (i.e. linux, your Ubuntu VM) and can be written in any language supported by that system.

If you don't know what to use, use C.

The program needs to be invoked as follows:

```
./mkboot -f floppyimage_to_modify kernel_to_put_on_floppy
```

Since we don't have our own file system yet, the `mkboot` tool can freely decide how and where to put the data on the floppy. However, make sure your code can handle non-contiguous or out of order mappings. In a real file system, files can become fragmented so the first part of a file might be in sectors 1000-4000 and the second part might be in 200-250. Make sure your tool can handle this.

Hints:

- The kernel size might not be a multiple of the sector size. In that case, the last sector containing the kernel should be padded with 0-bytes.
- Since LBA uses 64-bit numbers, the `stdint.h` header might be useful (`man stdint.h`).
- A template for the C program is available in the shared repository.
- Use the `hexdump` command to verify that everything was written to the correct location.

Task 2: Preparation for next step

The next step will be to switch the system to protected mode. However, there is a lot of background information required to fully understand how this works.

Please research the following topics. Make sure to keep a record of what you found (link + a small summary); You will need to hand in your findings.

- Detailed study about PROTECTED MODE for both 80286 and 80386 * see book chapters about memory management (main memory & virtual memory)
- Read about 386 registers: the wikipedia article about protected mode is very good
- Understand what LDT, GDT and IDT are.
- Research how we will organize our class project. For example, where and in which format should we keep documentation (including the information you've collected for this homework) so that it is well organized and easily accessible by everybody? How should we document the code?
- Understand how UNREAL mode works.

Task 3 for extra credit (optional)

Either one of the tasks below will get you extra credit;

Protected Mode

If, after completing task 2, you feel you have a good understanding of how to switch the processor to protected mode, you can update your bootloader load a protected mode 32 bit kernel. This doesn't affect your code loading the kernel into memory. The difference is that before you jump to the kernel, you switch to protected mode. Make sure interrupts are turned off. No page mapping needs to be enabled, but a proper code and data selector should be enabled.

Hints:

- http://prodebug.sourceforge.net/pmtut.html
- http://viralpatel.net/taj/tutorial/protectedmode.php

Elf Loader

So far, our resulting code has been in raw binary format, i.e. the layout on disk was exactly the same as the layout in memory. This is not very flexible, and cumbersome to work with. For example, it makes it hard to use gdb to debug our code.

The ELF format is a widely used standard for describing programs and libraries. It consists of a description of how these are stored on disk, and how they should be loaded into memory. Since for now, we will only be using statically linked programs (no external dependencies), loading an elf binary is easier than it looks!

Hints:

- http://codewiki.wikispaces.com/elf_load.nasm
- http://en.wikipedia.org/wiki/Executable_and_Linkable_Format

Resources

The following links contain useful information:

- Google
- Wikipedia

Images

- See the shared repository under `solutions/homework3` for images you can test with.