Homework: boot xv6

Submit your solutions before the beginning of the lecture (i.e., 1pm) on the due date mentioned on the schedule page to the submission web site.

Boot xv6

Login to Athena (e.g., ssh -X athena.dialup.mit.edu) and attach the course locker: (You must run this command every time you log in; or add it to your ~/.environment file.)

```
$ add -f 6.828
```

Fetch the xv6 source:

```
$ mkdir 6.828
$ cd 6.828
$ git clone git://github.com/mit-pdos/xv6-public.git
Cloning into 'xv6-public'...
...
$
```

Build xv6 on Athena:

```
$ cd xv6-public
$ make
...
gcc -O -nostdinc -I. -c bootmain.c
gcc -nostdinc -I. -c bootasm.S
ld -m    elf_i386 -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o
objdump -S bootblock.o > bootblock.asm
objcopy -S -O binary -j .text bootblock.o bootblock
...
$
```

If you are not using Athena for 6.828 JOS labs, but build on your own machine, see the instructions on the tools page. If you have a build infrastructure on your own machine for lab 1, then you should be able to use that infrastructure for building xv6 too.

Finding and breaking at an address

Find the address of `_start`, the entry point of the kernel:

```
$ nm kernel | grep _start
8010a48c D _binary_entryother_start
8010a460 D _binary_initcode_start
0010000c T _start
```

In this case, the address is `0010000c`.

Run the kernel inside QEMU GDB, setting a breakpoint at `_start` (i.e., the address you just found).

```
$ make qemu-gdb
...
<leave "make qemu-gdb" running, and in a new terminal, navigate to the same
directory and run the following. If you are trying this by logging into
athena.dialup.mit.edu, check the hostname to make sure that you are running
both the commands on the same physical machine.>
$ gdb
```

```
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
+ target remote localhost:26000
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0xf000,$0xe05b
0x0000fff0 in ?? ()
+ symbol-file kernel
(gdb) br * 0x0010000c
Breakpoint 1 at 0x10000c
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x10000c:    mov    %cr4,%eax

Breakpoint 1, 0x0010000c in ?? ()
(gdb)
```

The details of what you see are likely to differ from the above output, depending on the version of gdb you are using, but gdb should stop at the breakpoint, and it should be the above `mov` instruction. Your gdb may also complain that auto-loading isn't enabled. In that case, it will print instructions on how to enable auto-loading, and you should follow those instructions.

Exercise: What is on the stack?

While stopped at the above breakpoint, look at the registers and the stack contents:

```
(gdb) info reg
...
(gdb) x/24x $esp
...
(gdb)
```

Write a short (3-5 word) comment next to each non-zero value on the stack explaining what it is. Which part of the stack printout is actually the stack? (Hint: not all of it.)

You might find it convenient to consult the files bootasm.S, bootmain.c, and bootblock.asm (which contains the output of the compiler/assembler). The reference page has pointers to x86 assembly documentation, if you are wondering about the semantics of a particular instruction. Your goal is to understand and explain the contents of the stack that you saw above, just after entering the xv6 kernel. One way to achieve this would be to observe how and where the stack gets setup during early boot and then track the changes to the stack up until the point you are interested in. Here are some questions to help you along:

- Begin by restarting qemu and gdb, and set a break-point at 0x7c00, the start of the boot block (bootasm.S). Single step through the instructions (type `si` at the gdb

prompt). Where in bootasm.S is the stack pointer initialized? (Single step until you see an instruction that moves a value into `%esp`, the register for the stack pointer.)
- Single step through the call to `bootmain`; what is on the stack now?
- What do the first assembly instructions of bootmain do to the stack? Look for bootmain in bootblock.asm.
- Continue tracing via gdb (using breakpoints if necessary -- see hint below) and look for the call that changes `eip` to 0x10000c. What does that call do to the stack? (Hint: Think about what this call is trying to accomplish in the boot sequence and try to identify this point in bootmain.c, and the corresponding instruction in the bootmain code in bootblock.asm. This might help you set suitable breakpoints to speed things up.)

Submit: The output of `x/24x $esp` with the valid part of the stack marked, plus your comments, in a file named hwN.txt (where N is the homework number as listed on the schedule).