In-class: boot xv6

Boot xv6

Fetch the xv6 source:

```
$ mkdir 6.828
$ cd 6.828
$ git clone git://pdos.csail.mit.edu/xv6/xv6.git
Cloning into xv6...
...
$
```

Build xv6:

```
$ add -f 6.828
$ cd xv6
$ 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
8010b50c D _binary_entryother_start
8010b4e0 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
...
$ gdb
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
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".
+ 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.

Exercise: What is on the stack?

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 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. Here are some questions to help you along:

- Start by setting a break-point at 0x7c00, the start of the boot block (bootasm.S). Single step through the instructions (type si to the gdb prompt). Where in bootasm.S is the stack pointer initialized?
- 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.
- Look in bootmain in bootblock.asm for the call that changes `eip` to 0x10000c. What does that call do to the stack?

Submit: The output of `x/24x $esp` with the valid part of the stack marked plus your comments..

6.828 Operating System Engineering

Fall 2012