Homework 1

Due Date: Sept 25 at midnight.

In this assignment, you'll start getting familiar with xv6 by writing a couple simple programs that run in the xv6 OS.

As a prerequisite, make sure that you have followed the install instructions to get your build environment set up.

A common theme of the homework assignments is that we'll start off with xv6, and then add something or modify it in some way. This assignment is no exception. Start by getting a copy of xv6 using `git` (commands typed at the terminal, and their output, will be shown using a `monospace` font; the commands type will be indicated by a `$`):

```
$ git clone https://github.com/moyix/xv6-public.git
Cloning into 'xv6-public'...
remote: Counting objects: 4475, done.
remote: Compressing objects: 100% (2679/2679), done.
remote: Total 4475 (delta 1792), reused 4475 (delta 1792), pack-reused 0
Receiving objects: 100% (4475/4475), 11.66 MiB | 954.00 KiB/s, done.
Resolving deltas: 100% (1792/1792), done.
Checking connectivity... done.
```
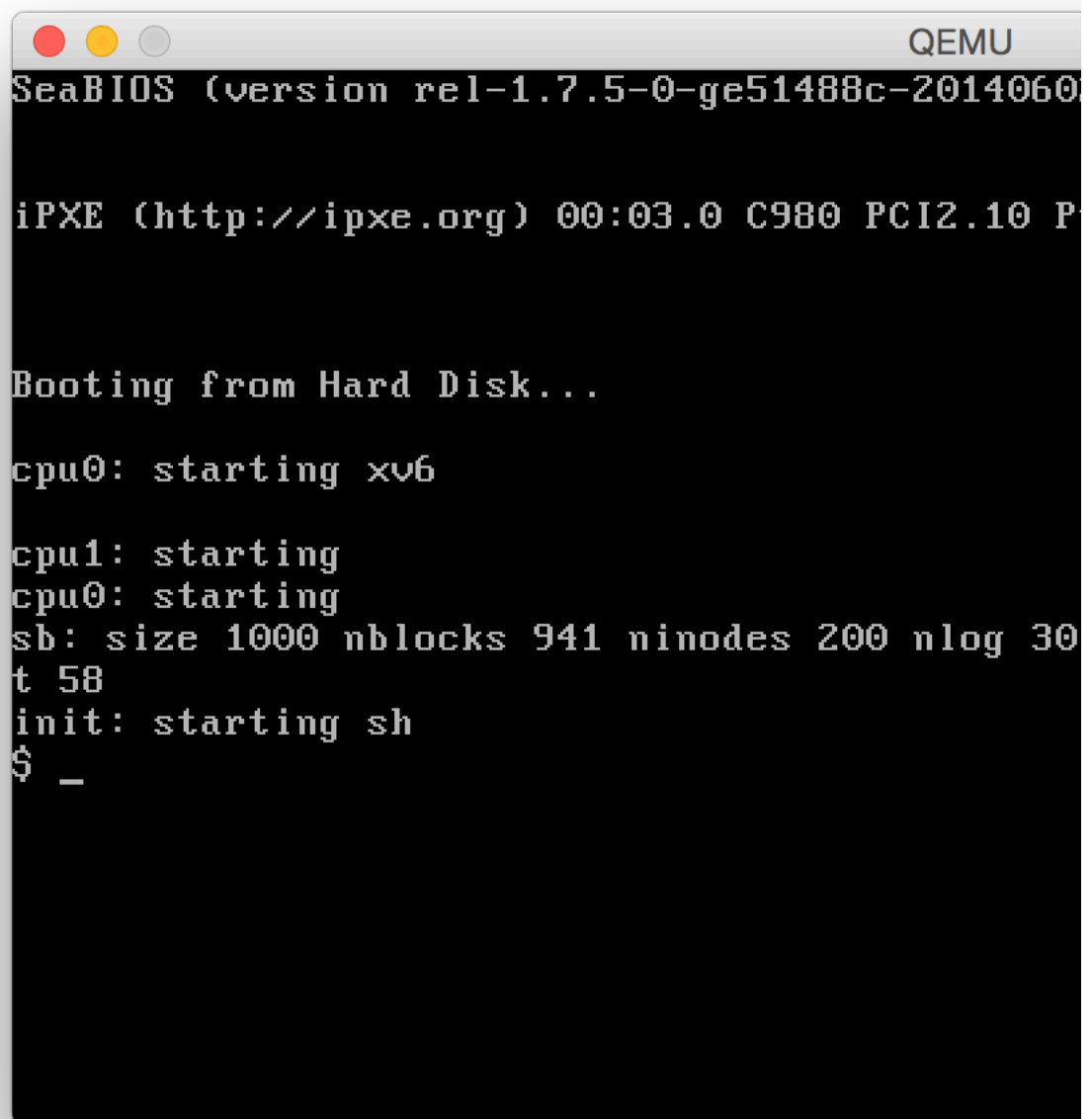
Make sure you can build and run xv6. To build the OS, use `cd` to change to the xv6 directory, and then run `make` to compile xv6:

```
$ cd xv6-public
$ make
```

Then, to run it inside of QEMU, you can do:

```
$ make qemu
```

QEMU should appear and show the xv6 command prompt, where you can run programs inside xv6. It will look something like:

```
SeaBIOS (version rel-1.7.5-0-ge51488c-2014060/

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 P

Booting from Hard Disk...

cpu0: starting xv6

cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30
t 58
init: starting sh
$ _
```
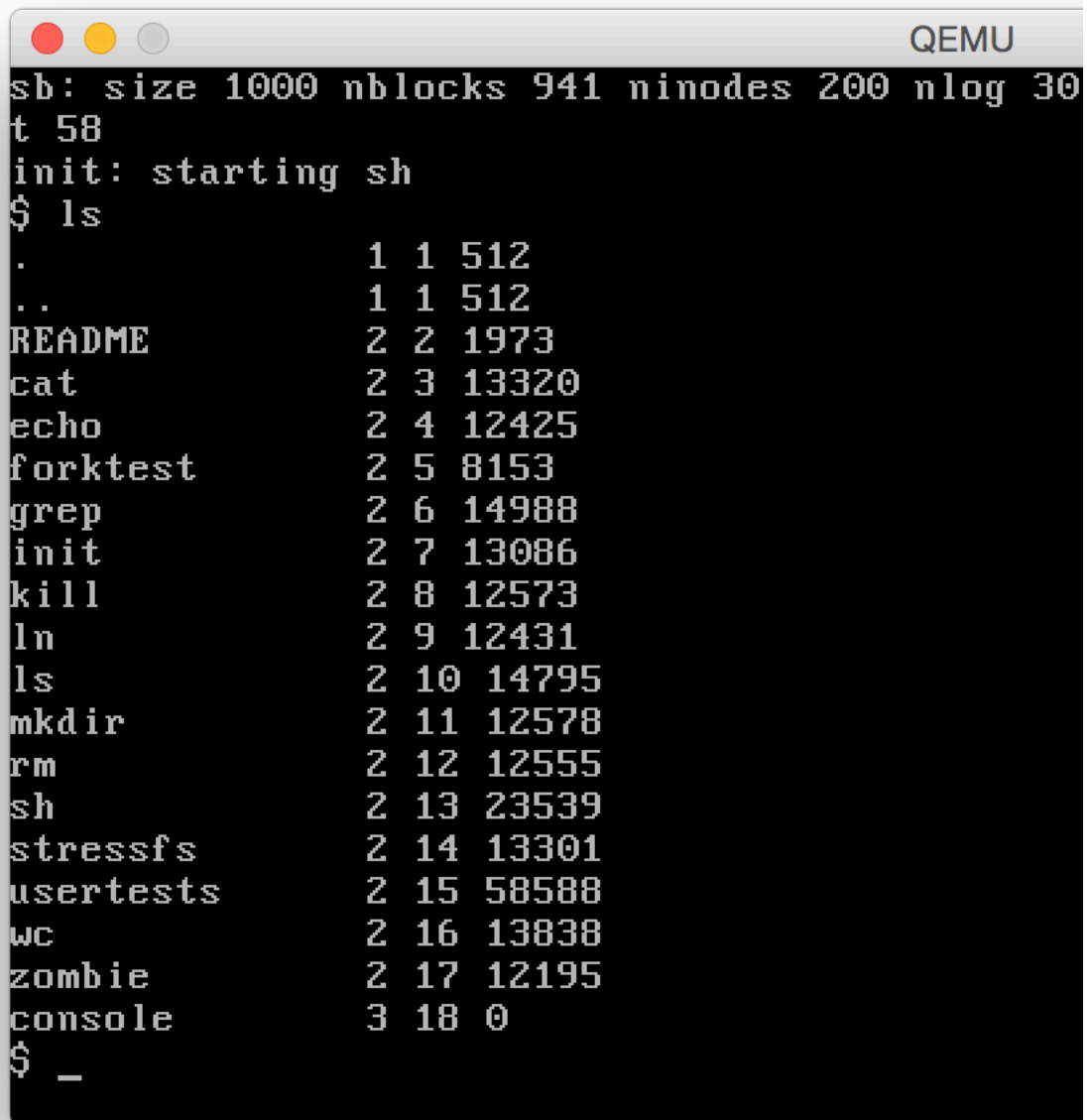
You can play around with running commands such as `ls`, `cat`, etc. by typing them into the QEMU window; for example, this is what it looks like when you run `ls` in xv6:

```
sb: size 1000 nblocks 941 ninodes 200 nlog 30
t 58
init: starting sh
$ ls
.                      1 1 512
..                     1 1 512
README                 2 2 1973
cat                    2 3 13320
echo                   2 4 12425
forktest               2 5 8153
grep                   2 6 14988
init                   2 7 13086
kill                   2 8 12573
ln                     2 9 12431
ls                     2 10 14795
mkdir                  2 11 12578
rm                     2 12 12555
sh                     2 13 23539
stressfs               2 14 13301
usertests              2 15 58588
wc                     2 16 13838
zombie                 2 17 12195
console                3 18 0
$ _
```

Part 1: Hello World (20 points)

Write a program for xv6 that, when run, prints "Hello world" to the xv6 console. This can be broken up into a few steps:

1. Create a file in the xv6 directory named `hello.c`
2. Put code you need to implement printing "Hello world" into `hello.c`
3. Edit the file `Makefile`, find the section `UPROGS` (which contains a list of programs to be built), and add a line to tell it to build your Hello World program. When you're done that portion of the `Makefile` should look like:

```
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _hello\
```

4. Run `make` to build xv6, including your new program (repeating steps 2 and 4 until you have compiling code)

5. Run `make qemu` to launch xv6, and then type `hello` in the QEMU window. You should see "Hello world" be printed out.

Of course step 2 is where the bulk of the work lies. You will find that many things are subtly different from the programming environments you've used before; for example, the `printf` function takes an extra argument that specifies where it should print to. This is because you're writing programs for a new operating system, and it doesn't have to follow the conventions of anything you've used before. To get a feel for how programs look in xv6, and how various APIs should be called, you can look at the source code for other utilities: `echo.c`, `cat.c`, `wc.c`, `ls.c`.

Hints:

1. In places where something asks for a file descriptor, you can use either an actual file descriptor (i.e., the return value of the `open` function), or one of the *standard I/O descriptors*: 0 is "standard input", 1 is "standard output", and 2 is "standard error". Writing to either 1 or 2 will result in something being printed to the screen.

2. The standard header files used by xv6 programs are `"types.h"` (to define some standard data types) and `"user.h"` (to declare some common functions). You can look at these files to see what code they contain and what functions they define.

A brief digression on IDEs and text editors

I do not have strong preferences as to *how* you create source code. I personally prefer to use a traditional text editor that can be run at the command line (specifically, `vim`; `emacs` is another worthy choice) for the purpose, but there are plenty of alternatives out there. On OS X, some may prefer to use XCode, others may prefer to use something like [TextMate](#) or [Sublime Text](#). In the Linux VM I have provided, `gedit` works fine. As long as you get a plain text file out of it with valid C syntax, you can choose whatever you like.

How you *compile* the code is another matter. The xv6 OS is set up to be built using `make`, which uses the rules defined in `Makefile` to compile the various pieces of xv6, and to allow you to run the code. The simplest way to build and run it is to use this system. Trying to coerce an IDE such as XCode into building xv6 is far more trouble than it's worth.

Part 2: Implementing the `head` command (50 points)

Write a program that prints the first 10 lines of its input. If a filename is provided on the command line (i.e., `head FILE`) then `head` should open it, read and print the first 10 lines, and then close it. If no filename is provided, `head` should read from standard input.

```
$ head README
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
but is implemented for a modern x86-based multiprocessor using ANSI C.

ACKNOWLEDGMENTS

xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (Peer
to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14,
2000)). See also http://pdos.csail.mit.edu/6.828/2014/xv6.html, which
provides pointers to on-line resources for v6.
```

You should also be able to invoke it without a file, and have it read from standard input. For example, you can use a *pipe* to direct the output of another xv6 command into `head`:

```
$ grep the README | head
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
xv6 borrows code from the following sources:
    JOS (asm.h, elf.h, mmu.h, bootasm.S, ide.c, console.c, and others)
    Plan 9 (entryother.S, mp.h, mp.c, lapic.c)
In addition, we are grateful for the bug reports and patches contributed by
The code in the files that constitute xv6 is
To run xv6, install the QEMU PC simulators.  To run in QEMU, run "make qemu".
To create a typeset version of the code, run "make xv6.pdf".  This
requires the "mpage" utility.  See http://www.mesa.nl/pub/mpage/.
```

The above command searches for all instances of the word `the` in the file `README`, and then prints the first 10 matching lines.

Hints:

1. Many aspects of this are similar to the `wc` program: both can read from standard input if no arguments are passed or read from a file if one is given on the command line. Reading its code will help you if you get stuck.

Part 3: Extending `head` (30 points)

The traditional UNIX `head` utility can print out a configurable number of lines from the start of a file. Implement this behavior in your version of `head`. The number of lines to be printed should be specified via a command line argument as `head -NUM FILE`, for example `head -3 README` to print the first 3 lines of the file README. The expected output of that command is:

```
$ head -3 README
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
but is implemented for a modern x86-based multiprocessor using ANSI C.
```

If the number of lines is not given (i.e., if the first argument does not start with `-`), the number of lines to be printed should default to 10 as in the previous part.

Hints:

1. You can convert a string to an integer with the `atoi` function.
2. You may want to use *pointer arithmetic* (discussed in class in Lecture 2) to get a string suitable for passing to `atoi`.

Submitting the Assignment

Submit `hello.c` and the completed `head.c` on NYU Classes, along with a file `partner.txt` that indicates who you worked with (if anyone).