Part 0: Intro to xv6

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

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/mit-pdos/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:
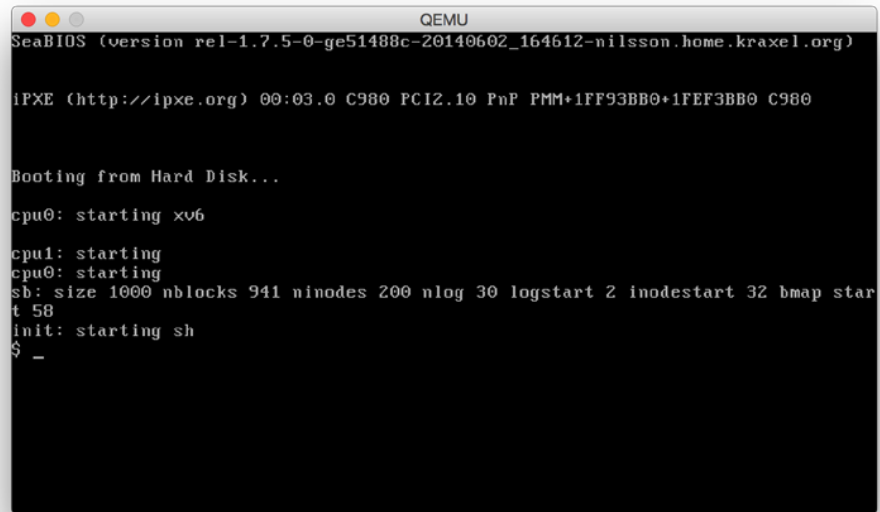
```
$ cd xv6-public
$ make
```

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:
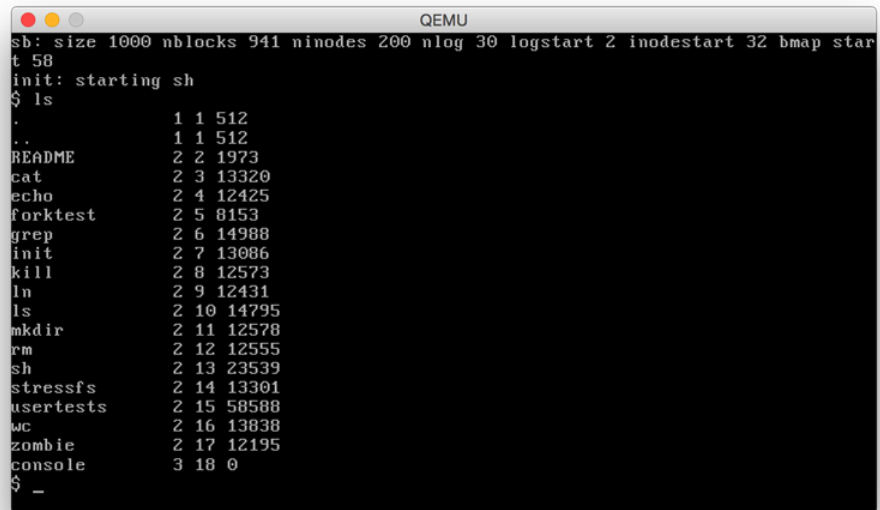
Figure 1: Starting xv6 in QEMU

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:

Figure 2: Running `ls` in xv6

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.

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

`uniq` is a Unix utility which, when fed a text file, outputs the file with adjacent identical lines collapsed to one. If a filename is provided on the command line (i.e., `uniq FILE`) then `uniq` should open it, read, filter out, print without repeated lines in this file, and then close it. If no filename is provided, `uniq` should read from standard input.

Here's an example of the basic usage of `uniq`:

```
$ cat example.txt
No. 1
No. 2
No. 2
No. 2
No. 3
No. 4
No. 5
No. 6
No. 6
No. 2
no. 2

$ uniq example.txt
No. 1
No. 2
No. 3
No. 4
No. 5
No. 6
No. 2
no. 2
```

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 `uniq`:

```
$ cat example.txt | uniq
No. 1
No. 2
No. 3
No. 4
No. 5
No. 6
No. 2
no. 2
```

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.

2. Still confused with `uniq`'s behavior? Use `man uniq` for help.

Part 3: Extending `uniq` (30 points)

The traditional UNIX `uniq` utility can do lots of things, such as:

- `-c`: count and group prefix lines by the number of occurrences

- `-d`: only print duplicate lines

- `-i`: ignore differences in case when comparing

Here, we are going to implement these three behaviors in your version of `uniq`. The expected output of these commands should be:

```
$ uniq -c example.txt
      1 No. 1
      3 No. 2
      1 No. 3
      1 No. 4
      1 No. 5
      2 No. 6
      1 No. 2
      1 no. 2

$ uniq -d example.txt
No. 2
No. 6

$ uniq -i example.txt
No. 1
No. 2
No. 3
No. 4
No. 5
No. 6
No. 2

$ uniq -c -i example.txt
      1 No. 1
      3 No. 2
      1 No. 3
      1 No. 4
      1 No. 5
      2 No. 6
      2 No. 2
```

Notice that "No. 2" should be the same as "no. 2" if `uniq` is not case-sensitive. Also, `-c` and `-d` won't appear at the same time.