Homework 1

Academic Honesty

Aside from the narrow exception for collaboration on homework, all work submitted in this course must be your own. Cheating and plagiarism will not be tolerated. If you have any questions about a specific case, please ask me. We will be checking for this!

NYU Poly's Policy on Academic Misconduct: http://engineering.nyu.edu/academics/code-of-conduct/academic-misconduct

Homework Notes :

General Notes:

- Read the assignment carefully, including what files to include.
- Don't assume limitations unless they are explicitly stated.
- Treat provided examples as just that, not exhaustive list of cases that should work.
- When in doubt regarding what needs to be done, ask. Another option is test it in the real UNIX operating system. Does it behave the same way?
- **TEST** your solutions, make sure they work. It's obvious when you didn't test the code.

Rubric :

Since we had some issues before on homework 1. Here are **some** of the things we know we will test, but these are not the **only** things we will test. Therefore make sure to test your program thoroughly and thoughtfully.

Total: 100 points

-50: Hello does not work

-10: No exit() at the end of hello.c

-50: uniq does not work

-10: uniq does not handle long lines on file (more than 512 characters)

-10: "cat example.txt | uniq" does not work

-10: uniq -c example.txt does not work.

-10: uniq -d example.txt does not work

-10: uniq -i example.txt does not work

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

Tools Install

As a prerequisite you will need to install the tools for XV6. In order to do that copy or clone the following github repo to your local machine.

https://github.com/LuvneeshM/os3224_vagrant

Once you clone the repo you will get a file called **instructions.pdf** in your machine. Follow those instructions to install the tools.

STOP!

Before **continuing** to the next section make sure that you follow instructions.pdf to the end.

Assignment

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. If you haven't, please follow the instructions to get your environment setup and to get `XV6` from `git`.

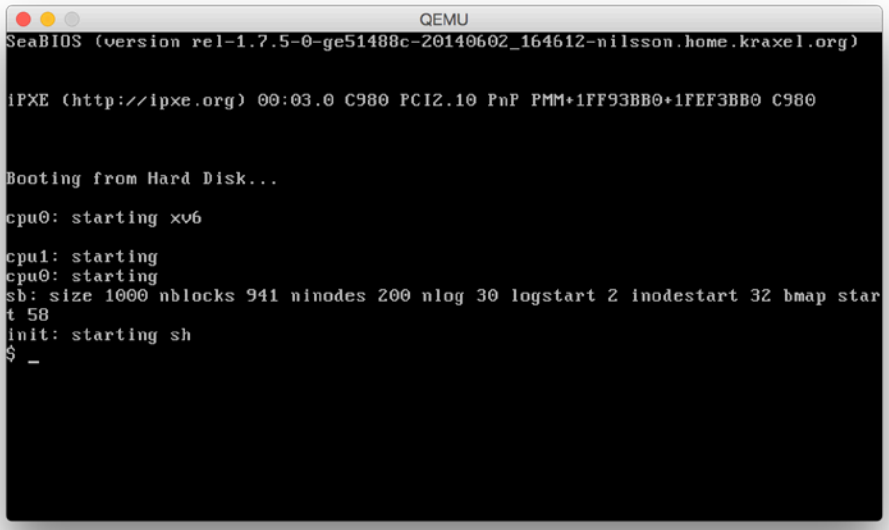
Make sure you can build and run xv6. To build the OS, use `cd` to change to the xv6 directory, and then run `make` to compile xv6:

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

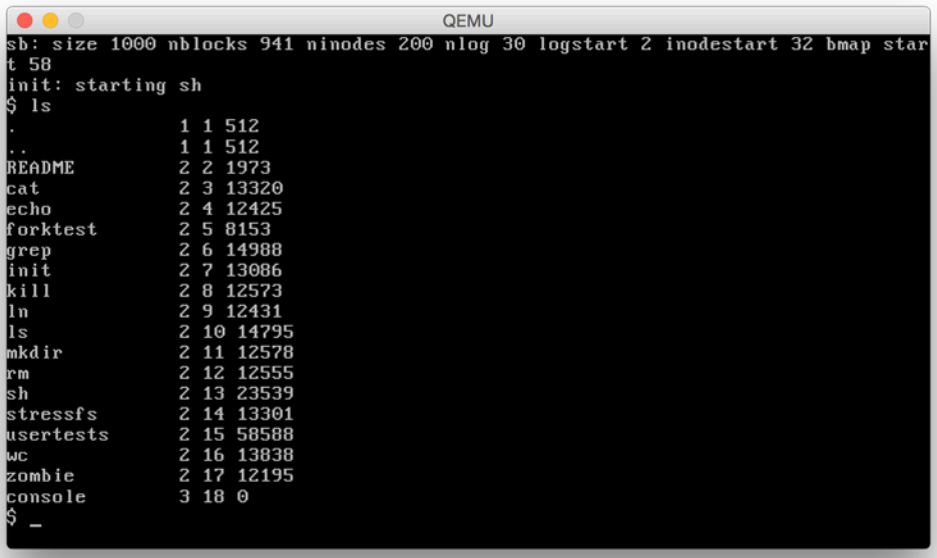
Then, to run it inside of QEMU, you can do:

```
$ make qemu
```

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



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:



You can exit XV6 by typing:

Ctrl-A X

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.

How to edit and compile code

As discussed in class, I do not have strong preferences as to *how* you create source code. I prefer and IDE but in some cases a traditional text editor that can be run at the command line such as `pico, vim or emacs` 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 a simple **'sed'** utility (20 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
  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.

Hints

1. You can use printf("%4d", number) to format the output.
2. There's always a space between the occurrence and content of that line.

Submitting the Assignment

Submit `hello.c` and the completed `uniq.c` on `NYU` Classes.