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

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

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

-70: sed does not work

-10: "cat README | sed" does not work

-40: "sed README" does not work

-10: Debug printf left in code

Due Date: Sept 16 at 11:55 pm

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

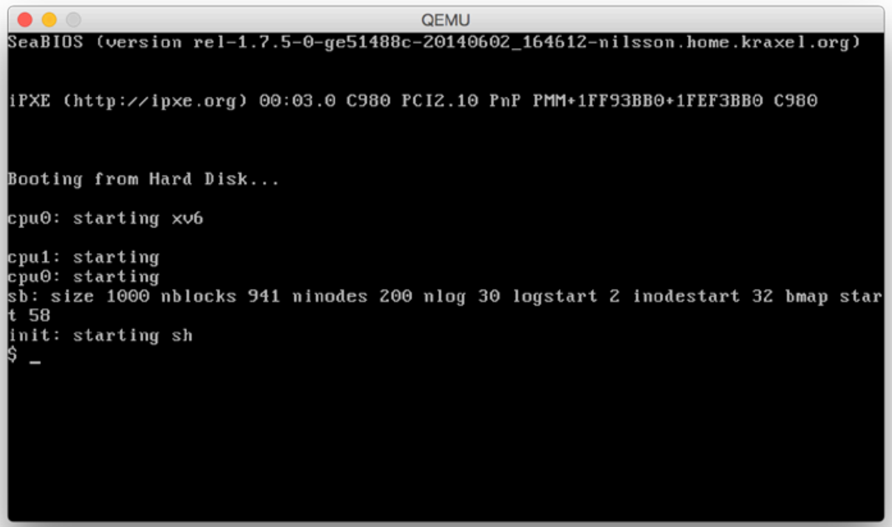
Make sure you can build and run xv6. To build the OS, use `cd` to change to the xv6 directory, and then run `make` to compile xv6:

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

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

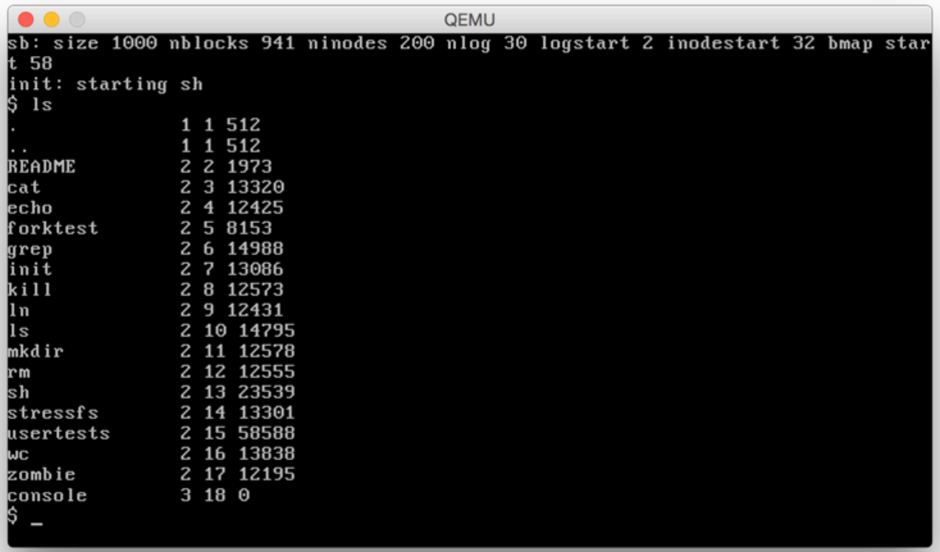
```
$ make qemu
```

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



```
SeaBIOS (version rel-1.7.5-0-ge51488c-20140602_164612-nilsson.home.kraxel.org)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF93BB0+1FEF3BB0 C980



Booting from Hard Disk...

cpu0: starting xv6

cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
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 logstart 2 inodestart 32 bmap star
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
$ _
```

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)

In the following parts of the assignment you will write a very simple version of `‘sed’`. `‘sed’` is a stream editor for filtering and transforming text that comes installed with most version of Unix. A stream editor performs basic text transformations on an input stream. You can read more about `‘sed’` by typing `‘man sed’`.

In part 2 you will write a version of `sed` that only does one thing: It counts the number of occurrences of the word "the" and then prints this count.
For example if you type the following it will print what follows the '>>'

```
$ sed README

>> 9
```

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 `sed` and it should print the same thing as before:

```
$ grep the README | sed

>> 9
```

The above command searches for all instances of the word `the` in the file `README,` and then prints them.

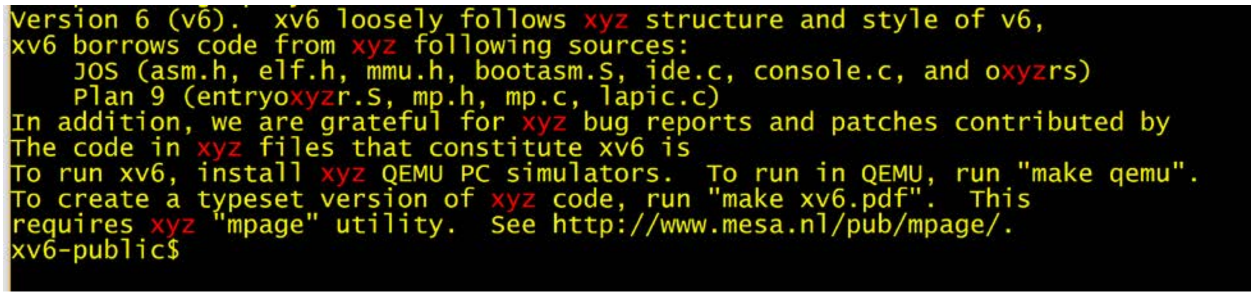
Hints:

1. Many aspects of this are similar to the `wc` program in XV6: 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 `sed` (30 points)

Now take this program to not only find all occurrences of the word `"the"` and print that number to standard output, but also now replace the word `"the"` with `"xyz".` If a filename is provided on the command line (i.e. `sed FILENAME`) then sed should open it, read and replace all occurrences of `"the"` with `"xyz"` and then close it. Afterwards it would then print the lines that would be modified as well as well as the number of occurrences. If no filename is provided, sed should read from standard input, print the input with the change and also the number of occurrences to standard output.

```
$ sed README


>> Found and Replaced 9 occurrences
```



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

Part 4: Final Extension to `sed` (30 points)

Extend the program to accept both the word/string/character to be replaced with its replacement via a command line argument as
    “`sed -TOBEREPLACED -REPLACEWITH FILENAME`”,

for example:

```
“sed –xv6 –x86 README”
```

Would swap all occurrences of xv6 with x86 in the file README and print them out. Note that in this case both the string to be replaced and the replace with string are the same size.

Part 5: Extra Credit Extension to `sed` (10 points)

Extend part 4 to accept strings of different sizes. This way you can call sed the following way:

```
“sed –xv6 –UNIX README”
```

Submitting the Assignment

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