CS537 Spring 2020, Project 1b

Administrivia

- **Due Date** by Feb 5, 2020 at 10:00 PM
- Questions: We will be using Piazza for all questions.
- Collaboration: The assignment has to be done by yourself. Copying code (from others) is considered cheating. Read this (http://pages.cs.wisc.edu/~remzi/Classes/537/Spring2018/dontcheat.html) for more info on what is OK and what is not. Please help us all have a good semester by not doing this.
- This project is to be done on the lab machines (https://csl.cs.wisc.edu/services/instructional-facilities).
- Some tests will be soon provided at *~cs537-1/tests/p1b*. Once they are available, read more about the tests, including how to run them, by executing the command `cat ~cs537-1/tests/p1b/README` on any lab machine. Note these test cases are not complete, and you are encouraged to create more on your own.

XV6 Basics

You will be using the current version of xv6. Begin by copying this directory **~cs537-1/xv6-sp20**. If, for development and testing, you would like to run xv6 in an environment other than the CSL instructional Linux cluster, you may need to set up additional software. You can read these instructions for the MacOS build environment (You'll still need to copy the version of xv6 in the previously mentioned directory). (link) (https://github.com/remzi-arpacidusseau/ostep-projects/blob/master/INSTALL-xv6.md) Note that we will run all of our tests and do our grading on the instructional Linux cluster so you should always ensure that the final code you handin works on those machines.

After you have obtained the source files, you can run **make qemu-nox** to compile all the code and run it using the QEMU emulator. Test out the unmodified code by running a few of the existing user-level applications, like ls and forktest.

To quit the emulator, type **Ctl-a x**. This means that you should press Control and A at the same time, release them, and then press x.

For additional information about xv6, we strongly encourage you to look through the code while reading this book (https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev11.pdf) by the xv6 authors. We are using a slightly different xv6 version, but most of the information is still accurate

Or, if you prefer to watch videos, the last ten minutes of the first video (https://www.youtube.com/watch?v=5H5esXbVkC8) plus a second video (https://www.youtube.com/watch?v=vR6z2QGcoo8&feature=youtu.be) from a previous year are relevant. You can also wait for the video from the discussion section this week Jan 30th!

Note that the version of xv6 we are using may be slightly different then that in the video. We always recommend that you look at the actual code yourself while either reading or watching (perhaps pausing the video as needed).

Assignment

We'll be doing kernel hacking projects in **xv6**, a port of a classic version of Unix to a modern processor, Intel's x86. It is a clean and small kernel.

This first project is intended to be a warmup, and thus relatively light. You will not write many lines of code for this project. Instead, a lot of your time will be spent learning where different routines are located in the existing source code.

Learning Objectives:

- Gain comfort looking through more substantial code bases written by others in which you do not need to understand every line
- Obtain familiarity with the xv6 code base in particular
- Learn how to add a system call to xv6
- Become familiar with a few of the data structures in xv6 (e.g., process table and file)
- Use the gdb debugger on xv6

Learning to Debug w/ GDB

The first part of this assignment it to learn and demonstrate your ability to use gdb to debug xv6 code. To do this we ask you to use gdb to show the integer value that **fdalloc** function returns the first time it is called after a process has been completely initiaized

To do this follow these steps:

1. Before you start, modify the ~/.gdbinit file to contain the directory where your xv6 directory is. For example if xv6-sp20 is in hour home directory, after your edit you should see something like

   ```
   # cat ~/.gdbinit
   add-auto-load-safe-path /afs/cs.wisc.edu/u/F/S/LOGIN/xv6-sp20/.gdbinit
   ```

   where F refers to the first letter of your CS username, S refers to the second letter of your CS username and LOGIN refers to your CS username.
2. In one window, type **make qemu-nox-gdb**.
3. In a second window on the same machine, cd to the **same directory** and type **gdb** (it will attach to the qemu process).
4. Once it is attached type **continue** in the gdb window. You will see xv6 finish its bootup process and you'll see it's shell prompt.
5. Interupt gdb and set a **breakpoint** at the **fdalloc** routine.
6. Continue **gdb**
7. In the xv6 shell, run the **stressfs** user application. Your gdb process should now have stopped in fdalloc.
8. Now, **step**(or probably **next**) through the C code until gdb reaches the point just before fdalloc() returns and **print** the value that will be returned (i.e. the value of fd).
9. Now immediately quit gdb and run **whoami** to display your login name.
10. Take a screenshot showing your gdb session with the returned value of fd printed and your login name displayed. Submit this screenshot to Canvas.

To sanity check your results, you should think about the value you expect fdalloc() to return in these circumstances to make sure you are looking at the right information. What is the first fd number returned after stdin, stdout, and stderr have been set up?

If gdb gives you the error message that fd has been optimized out and cannot be displayed, make sure that your Makefile uses the flag "-ggdb" instead of "-O2". Debugging is also a lot easier with a single CPU, so if isnt already set: in your **Makefile** find where the number of CPUS is set and change this to be 1.

If you get an error message saying that "... .gdbinit auto-loading has been declined by your `auto-load safe-path'..." then copy the path listed, open up the file ~/.gdbinit, paste the path, and try opening a fresh gdb instance. You may need to open a fresh window. For example the if the xv6-sp20 directory is in your home directory the contents of ~/.gdbinit would look like

```
# cat ~/.gdbinit
add-auto-load-safe-path /afs/cs.wisc.edu/u/F/S/LOGIN/xv6-sp20/.gdbinit
```

with F, S referring to the first letter, second letter of your CS username and and LOGIN referring to your CS username.

XV6 System Calls

For this assingment you'll be adding one system call to xv6:

- **int getfilenum(int pid)** which returns the number of files that the process identified by pid currently has open.

You must use the name of the system call EXACTLY as specified!

Implementation Details

The primary files you will want to examine in detail include **syscall.c**, **sysproc.c**, **proc.h**, and **proc.c**.

To add a system call, find some other very simple system call that also takes an integer parameter, like sys_kill, copy it in all the ways you think are needed, and modify it so it doesn't do anything and has the new name. Compile the code to see if you found everything you need to copy and change. You probably won't find everything the first time you try.

Then think about the changes that you will need to make so your system calls act like required.

- How will you find out the pid that has been passed to your system call? This is the same as what **sys_kill()** does.
- How will you find the data structures for the specified process? You'll need to look through the **ptable.proc** data structure to find the process with the matching pid.

To test your system call, you can make a simple user program that opens some files and calls your new system call. Similar to figuring out how to hook up your system call in all the right places, you can follow a user program such as ls and do the same.

We will also be releasing a simple test program and details about this will be posted on Piazza.

You may find the command **grep** helpful when you are trying to find all the places in the code base that you need to modify

Good luck! While the xv6 code base might seem intimidating at first, you only need to understand very small portions of it for this project. This project is very doable!

Handin

There are 2 steps

1. For your xv6 code, your handin directory is **~cs537-1/handin/LOGIN/p1b** where **LOGIN** is your CS login. Please create a subdirectory **~cs537-1/handin/LOGIN/p1b/src**. Copy all of your source files (but not .o files, please, or

binaries!) into this directory. A simple way to do this is to copy everything into the destination directory, then type "make" to make sure it builds, and then type "make clean" to remove unneeded files.
One way to do this is to navigate to your solution's working directory and execute the following command:

```
mkdir -p ~cs537-1/handin/LOGIN/p1b/src
```

```
cp -r . ~cs537-1/handin/LOGIN/p1b/src
```

When executing `ls -l` in the `~cs537-1/handin/LOGIN/p1b/src` directory the contents should be similar to what is shown below:

```
$ ls -l
total 28
-rw-r----- 1 <user-name> <user-name>  223 Feb 27  2012 FILES
drwxr-x--- 2 <user-name> <user-name> 2048 Jan 24 16:04 include/
drwxr-x--- 2 <user-name> <user-name> 6144 Jan 28 12:01 kernel/
-rw------- 1 <user-name> <user-name> 4823 Jan 23 14:51 Makefile
-rw-r----- 1 <user-name> <user-name> 1793 Feb 27  2012 README
drwxr-x--- 2 <user-name> <user-name> 2048 Jan 28 12:01 tools/
drwxr-x--- 2 <user-name> <user-name> 4096 Jan 28 12:01 user/
-rw-r----- 1 <user-name> <user-name>   22 Feb 27  2012 version
```

2. Upload your debugging **screenshot** to to Canvas

Acknowledgments

The assignment borrows content from assignment 2 of Prof. Andrea Arpaci-Dusseau's course in Fall 2019.