COMP 3500 Introduction to Operating Systems
Project 2 – An Introduction to OS/161

Short Version 1.5
Revised: Jan. 31, 2022

There should be no collaboration among students.

1. Introduction

The goal of assignment is to provide you with the opportunity to understand the structures of OS/161 and System/161. OS/161 is an operating system on which you will be working, whereas System/161 is a machine simulator on which the OS/161 runs.

2. OS/161 and System/161

The code you will be working on is comprised of two main components:

- OS/161: a simplified operating system developed by the Systems Research at Harvard group, at Harvard University. You will augment the functionality of the OS/161 in subsequent programming assignments.
- System/161: the machine simulator used to emulate the hardware on which your OS/161 will be running. The focus of this course is to design and implement operating systems rather than developing or simulating hardware. You may not need to change the machine simulator, but are required to modify or augment any portion of the OS/161 code that runs on it.

3. Tools for Project 2: Git, Gitlab, GDB, and Script

We will learn the following two programming tools that will make your assignments a lot easier. You can find detailed information pertinent to these two tools from the two pdf files: (1) Git-Cheat-Sheet.pdf and (2) gdb-Cheat-Sheet.pdf, which are available on Canvas.

- Git (Version Control System)
- GitLab (Web-based Git-Repository Manager)
- GDB (Gnu Debugger)
- The `script` Command

4. Getting Started

4.1. Setting up Your Account

Important! Please add to your PATH variable the directory containing all of the programs related to OS/161. If you are using bash as your shell you should add the following line near the end of the `~/.bashrc` file.

```
$export PATH=~/cs161/bin:$PATH
```

If you use `tcsh` as your shell, you should add the following line near the end of the `~/.cshrc` file:

```
$setenv PATH ~/cs161/bin:$PATH
```

The man page for `tcsh` includes a list of various things you can put in your prompt.

4.2. Getting the Distribution (submit)

You have to download, build and install the distributions of the OS/161, System/161 MIPS emulator, and the OS161 toolchain.

1) tool chain: `cs161-binutils-1.5.tgz`
2) cross compiler: `cs161-gcc-1.5.tgz`
3) special gdb: `cs161-gdb-1.5.tgz`
4) sys161 MIPS emulator: `sys161-1.14.tar.gz`
5) OS/161: `os161-1.10.tar.gz`

You may follow the instructions below to build the tool chain, cross compiler, special gdb, and sys161.

4.2.1 Build the tool chain

```
%export CFLAGS="-g -O2 -Wno-error"
$cd ~/cs161
$tar vfzx cs161-binutils-1.5.tgz
$cd cs161-binutils-1.5
$./toolbuild.sh
```

4.2.2 Build the cross compiler

```
$cd ~/cs161
$tar vfzx cs161-gcc-1.5.tgz
$cd cs161-gcc-1.5
$./toolbuild.sh
```

4.2.3 Build the special gdb

```
$cd ~/cs161
$tar vfzx cs161-gdb-1.5.tgz
$cd cs161-gdb-1.5
$./toolbuild.sh
```

4.2.4 Build the sys161 emulator

```
$cd ~/cs161
$tar vfzx sys161-1.14.tar.gz
$cd sys161-1.14
$./configure mipseb
$make
$make install
```

4.3. Scripting Your Session (submit)

4.3.1 Script the following session using the `script` command.

4.3.2 Make a directory in which you will do all your programming assignments.

```
$mkdir ~/cs161
$mkdir ~/cs161/asst0
$cd cs161
```

4.3.3 Unpack the OS/161 distribution by typing:

```
$tar vfzx os161-1.10.tar.gz
```

4.3.4 The above step creates a directory named `os161-1.10`. Rename your OS/161 source tree to just `os161`.

```
$mv os161-1.10 os161
```

4.3.5 End your script session by typing exit or by pressing Ctrl-D. Rename your typescript file to be `setup.script`.

```
$mv typescript ~/cs161/asst0/setup.script
```

4.4 Setting up your GitLab or GitHub repository

Important! You may opt for making use of GitHub rather than GitLab to maintain your remote repository.

4.4.1 Use your Engineering account (user name and password) to login to the GitLab website here: https://gitlab.eng.auburn.edu

4.4.2 Create a new empty project; the project name should be os161. The visibility level must be *private*, because you don't want your code to be stolen.

4.4.3 The project created on GitLab will be the remote repository of your project 2. Follow Step 4.5 to create a local repository to be connected to the remote repository constructed in Step 4.4 (i.e., this step).

4.5 Setting up your Git repository (submit)

Script the following session using the `script` command.

4.5.1 Go to your local workspace:

```
$cd ~/cs161/os161
```

4.5.2 Create your Git local repository; connect the local Git repository with your remote one at GitLab; commit all the source code files from your workspace into your local; and push the source code from the local repository to the remote repository at GitLab.

```
$git init
$git remote add origin
https://gitlab.eng.auburn.edu/usr_name/os161.git
$git add .
$git commit -m "Initial OS161 commit"
$git push -u origin master
```

4.5.3 Go to your GitLab project webpage and check your remote repository. The URL of the project webpage is listed below, where `usr_name` is your Engineering user name:
https://gitlab.eng.auburn.edu/usr_name/os161

4.5.4 Now, you can remove the source tree in your local workspace (i.e., `~/cs161/os161`).

```
$cd ~/cs161
$rm -rf os161
```

Don't worry about your deleted local workspace, because you have backup the source-code tree in your remote GitLab repository. In the next step, you clone a copy of the source tree that is yours to work on.

4.5.5 Now, clone a source-code tree from the remote GitLab repository into your local machine where the new workspace is the `~/cs161/src` directory. `GitLab` will ask you to entire your user name and password when you attempt to clone the project.

```
$git clone
https://gitlab.eng.auburn.edu/usr_name/os161.git src
```

4.5.6 End your script session. Rename your script output to `gitinit.script`.

```
$mv typescript ~/cs161/asst0/gitinit.script
```

5. Code Reading (submit)

The goal of this exercise is to understand our base system. You should aim at understanding how it all fits together so that you can make intelligent design decisions when you approach future assignments. The questions below (which appear in red text) are not meant to be tricky--most of the answers can be found in comments in the OS/161 source, though you may have to look elsewhere (such as Silberschatz et al.) for some background information. Place your answers to the following questions in a file called `~/cs161/asst0/code-reading.txt`.

1. Which register number is used for the stack pointer (sp) in OS/161?
2. What bus/busses does OS/161 support?
3. What is the difference between `splhigh` and `spl0`?
4. Why do we use typedefs like u_int32_t instead of simply saying "int"?
5. What does splx return?
6. What is the highest interrupt level?
7. How frequently are hardclock interrupts generated?
8. What functions comprise the standard interface to a VFS device?
9. How many characters are allowed in a volume name?
10. How many direct blocks does an SFS file have?
11. What is the standard interface to a file system (i.e., what functions must you implement to implement a new file system)?
12. What function puts a thread to sleep?
13. How large are OS/161 pids?
14. What operations can you do on a vnode?
15. What is the maximum path length in OS/161?
16. What is the system call number for a reboot?
17. Where is STDIN_FILENO defined?
18. Is it OK to initialize the thread system before the scheduler? Why or why not?
19. What is a zombie?
20. How large is the initial run queue?
21. What does a device name in OS/161 look like?
22. What does a raw device name in OS/161 look like?
23. What lock protects the vnode reference count?
24. What device types are currently supported?

6. Building a Kernel (submit)

Warning! If you haven't configured your `PATH` variable, you must add the following directory into the `PATH` variable. If you are using bash as your shell you should add the following line near the end of the `~/.bashrc` file.

```
export PATH=~/cs161/bin:$PATH
```

6.1 Configure your tree for the machine on which you are working. We assume that you work in the directory `~/cs161`. Please note that if you intend to work in a directory that's not `~/cs161` (which you will be doing when you test your later submissions), you will have to use the `–ostree` option to specify a directory in which you are working. `./configure`

`-help` explains the other options.

```
$cd ~/cs161/src
$./configure
```

6.2 Configure a kernel named ASST0.

```
$cd ~/cs161/src/kern/conf
$./config ASST0
```

6.3 Build the ASST0 kernel.

```
$cd ../compile/ASST0
$make depend
$make
```

6.4 Install the ASST0 kernel.

```
$make install
```

6.5 Now also build the user level utilties.

```
$cd ~/cs161/src
$make
```

6.7. End your script session. Rename your script output to `build.script`.

```
$mv typescript ~/cs161/asst0/build.script
```

7. Running your kernel (submit)

Download the file `sys161.conf` from Canvas and place it in your OS/161 root directory (`~/cs161/root`). Script the following session.

7.1 Change into your root directory.

```
$cd ~/cs161/root
```

7.2 Run the machine simulator on your operating system.

```
$./sys161 kernel
```

7.3 At the prompt, type `p /sbin/poweroff <return>`. This tells the kernel to run the "`poweroff`" program that shuts the system down.

7.4 End your script session. Rename your script output to `run.script`.

```
$mv typescript ~/cs161/asst0/run.script
```

8. Practice modifying your kernel (submit)

If you haven't configured your `PATH` variable, you must add the following directory into the `PATH` variable. Script the following session.

8.1 Create a file called `~/cs161/src/kern/main/hello.c`

8.2 Stage the above new file to your local `Git` repository:

```
%git add ~/cs161/src/kern/main/hello.c
```

8.3 In this file, write a function called `hello()` that uses `kprintf()` to print "`Hello World\n`".

8.4 Edit `kern/main/main.c` and add a call in a suitable place to `hello()`.

8.5 Edit `kern/conf/conf.kern` using the following commands

```
cd ~/cs161/src/kern/conf
vi conf.kern
/main.c  #search main.c in conf.kern see also line 374
Add the following line:
file  main/hello.c
```

8.6 You must reconfig, and then rebuild (see Section 6 for details)

8.7 Make sure that your new kernel runs and displays the new message. Once your kernel builds, script a session demonstrating the config and build of your modified kernel (see also Sections 6 and 7). Call the output of this script session `newbuild.script`.

```
$mv typescript ~/cs161/asst0/newbuild.script
```

9. Using GDB (submit)

You will require two windows for the following portion.

9.1 Script the following GDB session (that is, you needn't script the session in the run window, only the session in the debug window). Be sure both your run window and your debug window are on the same machine.

9.2 Run the kernel in GDB by first running the kernel and then attaching to it from GDB.

```
#In the run window
  $cd ~/cs161/root
  $./sys161 -w kernel

#In the debug window
  cd ~/cs161/root
  $cs161-gdb kernel
  (gdb) target remote unix:.sockets/gdb
  (gdb) break menu
  (gdb) c
  #gdb will stop at menu()

  (gdb) where

  #displays a nice back trace
  (gdb) detach
  (gdb) quit
```

9.3 End your script session. Rename your script output to gdb.script.

```
$mv typescript ~/cs161/asst0/gdb.script
```

10. Practice with Git (submit)

Create a script of the following session (the script should contain everything except the editing sessions; perform those in a different window so they don't appear in the script). Call this file

git-use.script.

10.1 Edit the file `kern/main/main.c`. Add a comment with your name in it.

10.2 Execute

```
$git diff
```

to display the differences in your version of this file.

10.3 Now commit your changes using `git commit`.

10.4 Remove the first 100 lines of `main.c`.

10.5 Try to build your kernel (this ought to fail).

10.6 Realize the error of your ways and get back a good copy of the file.

```
$rm main.c
$git checkout main.c
```

10.7 Try to build your tree again (see Section 6).

10.8 Now, examine the DEBUG macro in `src/kern/include/lib.h`. Based on your earlier reading of the operating system, add ten useful debugging messages to the source code of your operating system os161.

A sample source code is given below:

```
DEBUG(DB_VM, "VM free pages: %u\n", free_pages);
```

In the above code, `DB_VM` is a flag defined in `src/kern/include/lib.h`
If the debug DB_VM flag is set, the debug message will be printed on the console.

10.9 Now, display the locations where you inserted these DEBUG statements by doing a diff.

```
$cd ~/cs161/src
$git diff
```

10.10 **Important!** Finally, you should create a release using "`git archive`".

```
$cd ~/cs161/src
$git commit -m "project 2"
$git tag asst0-end
$git tag
$git show
$git push --tags
$git archive -o ../asst0/project2_source.tgz HEAD
$cd ~/cs161
$tar vfcz asst0.tgz asst0
```

Note: The `git archive` command creates a tarball named `project2_source.tgz` in the `~/cs161/asst0` directory. The `tar vfcz` command (i.e., the last command) aims to build a final tarball by compressing `project2_source.tgz` along with the other eight (8) script files in the `~/cs161/asst0` directory.

11. Deliverables

Your `asst0` directory, which you tarred and compressed above, should contain everything you

need to submit, specifically:

1) `setup.script`
2) `gitinit.script`
3) `code-reading.txt`
4) `build.script`
5) `run.script`
6) `newbuild.script`
7) `gdb.script`
8) `git-use.script`
9) `project2_source.tgz`

Now, submit your tarred and compressed file named `asst0.tgz` through Canvas. You must submit your single compressed file through Canvas (no e-mail submission is accepted.