COMP3231/9201/3891/9283 Operating Systems 2020/T1

UNSW

Assignment 0: Code Walkthrough

Note: this part is non-assessable. Feel free to discuss them with fellow students.

This is probably the first time most of you will attempt to understand, and carefully modify, a large body of code that you did not write yourself. It is imperative that you take the time to get an understanding of the overall structure of the code, and roughly what each part of the code does.

This non-assessable, code reading component of this assignment aims to guide you through the code base to help you start to comprehend its contents, identify what functionality is implemented where, and be able to make intelligent decisions on how to modify the code base to achieve the goals of the assignments.

You don't need to understand every line of code, but you should have a rough idea of what some directories or files contain.

Invest a little time now to gain some practice navigating around the code base. Now is probably the least busiest part of the semester for you.

Invest time in setting up a source code browser. The ease of navigating the code has a large influence on your ability to comprehend how it fits together. You can avoid knowing about the directory structure and location of source code files with the right tools that can navigate directly to a function or data structure definition by name or just clicking. Some suggestions for tools are Eclipse (see our set up guide), cscope (emacs or vim), etags (emacs) , ctags (vim), global (emacs or vim) or similar. We provide a rough (potentially out of date) guide to setting up Eclipse on the wiki.

The top-level directory

The `asst0-src` directory contains the top-level directory of the OS/161. It contains a few files, and subdirectories containing distinct parts of OS/161. The files are:

- **Makefile** this makefile builds the OS/161 distribution, including all the provided utilities. It does not build the operating system kernel.
- **configure**: this is a configuration script, similar to autoconf, but not generated by autoconf. You shouldn't need to understand or tamper with it.
- **defs.mk**: this file is generated by running `./configure`. Unless something goes wrong, you shouldn't need to do anything with it.

`asst0-src` contains the following directories:

- **kern**: contains the sources to the OS/161 kernel itself. We will cover this in more detail later.
- **userland**: sources for the user-level utilities and tests.
- **common**: code shared between user-level and the kernel.
- **mk**: build system support code.
- **build**: temporary output directory for user-level binaries.
- **man**: documentation for user-level utilities and APIs.

Your focus during this code walk through should be on the kernel sources. You won't need a detailed understanding of the utilities in `userland`, however broad understanding of how they work and where things are is useful.

The Kern Subdirectory

This directory and its subdirectories are where most (if not all) of the action takes place. The only file in this directory is a `Makefile`. This Makefile only installs various header files. It does not actually build anything.

We will now examine the various subdirectories in detail. **Take time to explore the code and answer the questions.**

kern/arch

This directory contains architecture- and platform-dependent code, which means it is specific to a particular type of CPU that OS/161 runs on. Different machine architectures have their own specific architecture-dependent directory. Currently, there is only one supported architecture, which is **mips**, and one supported machine, **sys161**.

kern/arch/mips/conf

`conf.arch`: This tells the kernel config script where to find the machine-specific, low-level functions it needs.

Question 1: What is the vm system called that is configured for assignment 0?

kern/arch/mips/include

These files are include files for the machine-specific constants and functions.

kern/arch/mips/locore
kern/arch/mips/syscall
kern/arch/mips/thread
kern/arch/mips/vm

These are the low-level functions the kernel needs that are machine-dependent.

Question 2. Which register number is used for the stack pointer (sp) in OS/161?
Question 3. What bus/busses does OS/161 support?

Question 4. Why do we use typedefs like `uint32_t` instead of simply saying "`int`"?
Question 5. What function is called when user-level code generates a fatal fault?

kern/compile

This is where you build kernels. In the compile directory, you will find one subdirectory for each kernel you want to build. In a real installation, these will often correspond to things like a debug build, a profiling build, etc. In our world, each build directory will correspond to a programming assignment, e.g., ASST1, ASST2, etc. These directories are created when you configure a kernel (described in the next section). This directory and build organisation is typical of UNIX installations and is not necessarily universal across all operating systems.

kern/conf

`config` is a shell script that takes a config file, like ASST1, and creates the corresponding build directory. Later (**not now**), in order to build a kernel, you will do the following:

```
% cd kern/conf
% ./config ASST0
% cd ../compile/ASST0
% bmake depend
% bmake
```

This will create the ASST0 build directory and then actually build a kernel in it. **Note that you should specify the complete pathname `./config` when you configure OS/161. If you omit the `./`, you may end up running the configuration command for the system on which you are building OS/161, and that is almost guaranteed to produce rather strange results!**

kern/include

These are the include files that the kernel needs. The kern subdirectory contains include files that are visible not only to the operating system itself, but also to user-level programs.

Question 6. How frequently are hardclock interrupts generated?
Question 7. How many characters are allowed in an SFS volume name?
Question 8. How many direct blocks does an SFS file have?
Question 9. What is the standard interface to a file system (i.e., what functions must you implement to implement a new file system)?
Question 10. What function puts a thread to sleep?
Question 11. How large are OS/161 pids?
Question 12. What operations can you do on a vnode?
Question 13. What is the maximum path length in OS/161?
Question 14. What is the system call number for a reboot?
Question 15. Where is STDIN_FILENO defined?

kern/main

This is where the kernel is initialised and where the kernel main function is implemented.

Question 16. What does `kmain()` do?

kern/thread

Threads are the fundamental abstraction on which the kernel is built.

Question 17. What is the difference between `splhigh` and `spl0`?
Question 18. What does splx return?
Question 19. What is a zombie?

kern/lib

These are library routines used throughout the kernel, e.g., string manipulation, bitmaps, console management, etc.

kern/vm

Virtual memory would be mostly implemented in here. Currently it contains basic address-space management functionality and the kernel malloc implementation.

kern/fs

The file system implementations. OS/161 currently implements one file system called **sfs**.

kern/fs/sfs

This is the simple file system that OS/161 contains by default. You may augment this file system as part of a future assignment, so we'll ask you questions about it then.

kern/vfs

This is the file-system independent layer (vfs stands for "Virtual File System"). It establishes a framework into which you can add new file systems easily. You will want to review vfs.h and vnode.h before looking at this directory.

Question 20. What does a device name in OS/161 look like?
Question 21. What does a raw device name in OS/161 look like?
Question 22. What lock protects the vnode reference count?
Question 23. What device types are currently supported?

kern/dev

This is where all the low level device management code is stored. You can safely ignore most of this directory.

This concludes the non-assessable reading component of the assignment. Feel free to discuss your answers with fellow students or your tutor. Basically, anybody who will listen :-)

Answers to these questions are available on the wiki on the class website.

Page last modified: 7:57pm on Sunday, 9th of February, 2020

Screen Version

CRICOS Provider Number: 00098G