CS350: Assignment 0 – Basic Setup

Tavian Barnes and Emil Tsalapatis
(with edits by various CS350 instructors)

1 Overview

For this course, you will be writing code for a small operating system named CastorOS. CastorOS is an educational OS that runs on real hardware or in a virtual machine [1]. During the course of the assignments, you will write code to boot and run CastorOS, and then add system calls to it to implement fundamental OS functionality for loading programs, managing processes and accessing the file system.

This assignment will familiarize you with downloading, editing, and submitting the CastorOS code, preparing you for future assignments. The code you downloaded does not compile in its current form; it is your job to find the error and make a simple change so that it builds successfully. Once you have done that, the OS will be able to start booting, but it will not finish booting successfully yet. Getting the boot process to succeed will be your task for Assignment 1.

2 One-time Setup

2.1 What You Need To Get Started

This assignment assumes you are using a terminal window connected to our undergraduate environment, linux.student.cs.uwaterloo.ca, via secure shell (SSH). This is the only officially supported way to complete the assignments. If you would like to set this up on your own machine, see Section 4 for some suggestions on how to complete the assignment in a Windows or macOS environment.

Besides this document, you need two items before you can start:

1. an email from our submission server with the subject line "CS350 Submission Magic" and
2. our submission system client, called `client.py`.

These may not be available at the start of the course. The instructors or IAs will notify you when they become available, usually via a lecture announcement and a Piazza post.

Typically you would obtain the submission client from the Assignments Information section of our course website, `https://student.cs.uwaterloo.ca/~cs350`.

If you want to download it while using a terminal window, rather than a browser, use the `wget` command. For example

```
wget https://cs350.rcs.uwaterloo.ca/download/client.py
```

The directory you download it into should initially be empty. We will refer to this as your *client directory*.

2.2 Setting Up and Verifying the Submission Client

To authenticate with the submission server, you will need to use your username and a magic string. Your username is your 8-character name that you use to login to the undergrad environment. You will receive the magic string by email from cs350-noreply@uwaterloo.ca. That string is specific for you. Do not share it with anyone else.

Edit client.py directly to set the STUDENT variable to your username, and the MAGIC variable to your magic string. To test whether the server is up, ping it with the following command:

```
python client.py ping
```

and the system should respond with

```
CS350 Server is active
```

The ping command confirms that the submission server is accessible from your machine. If the server is active, the following command ensures you have the proper credentials.

```
python client.py status -a asst0
```

If you have properly added your credentials to client.py, the server will respond with your last active submission and your grades, which in this case will be:

```
Last pending submission: None
Last grades: None
```

In the email you receive from our server and in the client.py script, your username is referred to as STUDENT. In the CastorOS source code, it is referred to as "yourname" or "USERNAME."

2.3 Downloading CastorOS

Next, download the latest version of the CastorOS source into the client directory using:

```
python client.py download
```

This command requests CastorOS's source from the submission server and places it in the current directory. If successful, you should now have the file castoros-latest.tar.gz in your client directory.

Use the linux command `tar` (roughly the equivalent of unzipping a zip file) to unpack the source into a castoros directory, in which you will develop your solutions.

```
tar zxvf castoros-latest.tar.gz
```

If successful, when you execute the command `ls`, you should see the following in your client directory:

```
castoros  castoros-latest.tar.gz  client.py
```

Enter the castoros directory with the command `cd castoros` and you should see the following:

```
AUTHORS  bin  build  docs  include  lib  LICENSE  pxelinux
README  release  sbin  SConstruct  sys  tags  tests
```

This directory, which we will call the *castoros source directory*, includes the userspace, kernel, and tools for the OS. It holds the different userspace components and the kernel in separate directories. The kernel is in the sys/ directory and includes device drivers, system calls and core subsystems. Userspace programs are in the bin/ and sbin/ directories. These programs

use the system API defined within the headers in the directory include. The lib directory holds userspace shared libraries like libc that implement the userspace part of this API and often interface with the kernel. The build directory is initially empty and serves as a destination for the compiled kernel binary and disk image.

2.4 Setting up Scons

The assignments use SCons [2] (a Software CONStruction tool) to compile CastorOS and create a runnable image. This tool is similar to `make` but uses Python rather than a custom language. The script that it uses (equivalent to a makefile) is called SConstruct and it is located in the castoros source directory. If you are using linux.student.cs.uwaterloo.ca, SCons is already installed for you. Otherwise, you may have to install it yourself.

Compiling the code is a two-step process handled automatically by SCons. (1) C code is compiled into an intermediate representation by Clang [3]. Clang performs parsing and type checking. (2) LLVM [4] (Low Level Virtual Machine) then optimizes this intermediate representation and generates machine code.

OS images must conform to a very precise layout specification. However, different compilers, or even versions of the same compiler, generate different layouts for the same code and build arguments. LLVM 15 has been confirmed to produce a correct image so we will be using it for all assignments. Ensure that SCons uses LLVM 15 by creating a file called Local.sc in the castoros source directory which sets the exact compiler (CC) and linker (LINK) to use. Your Local.sc file should look exactly like the following

```
CC="clang-15"
LINK="clang-15"
```

Note that each line in Local.sc ends in a newline.

The above configuration will work on the linux.student.cs environment. The Clang command may have a different name under different OSes (for example, clang15). Please consult your OS's documentation for more details if running clang-15 fails.

To ensure everything is working, execute the following command in the castoros source directory.

```
scons
```

The output it generates starts off with

```
scons: Reading SConscript files ...
Checking whether the C compiler works... yes
scons: done reading SConscript files.
scons: Building targets ...
```

and after over 100 lines ends with

```
scons: building terminated because of errors.
```

3 Editing, Testing and Submitting Your Code

These steps will be done each time you plan to edit and submit your code. For Assignment 0, there is a single error in your code, fix it so your code compiles and then submit it.

3.1 1. Edit Your Code

Edit your code to fix the error (A0) or add the required features (A1–A3).

3.2 2. Build Using SCons

In the castoros source directory (where the SConstruct file is), execute the following command

```
scons
```

If scons is successful, your output should end with the line

```
scons: done building targets.
```

You can verify that the proper files were created using the following command:

```
ls -lh build/bootdisk.img build/sys/castor
```

which should result in output similar to the following

```
-rw-r----- 1 user users 128M Sep 11 16:24 build/bootdisk.img
-rwxr-x--- 1 user users 189K Sep 11 16:24 build/sys/castor
```

3.3 Testing Your Code

For Assignment 0, test your code by trying to boot up CastorOS. Future assignments will give you other tests to do.

We start up CastorOS by passing the outputs of the compilation process to QEMU. QEMU (Quick EMUlator) is used here to emulate an x64 processor, allowing you to run your OS in the undergraduate environment. QEMU uses the disk image to emulate a hard disk. The disk image holds a file system with the userspace utilities, configuration files and user data. The kernel image only includes the kernel and it is separate because it must be passed directly to QEMU for the machine to boot.

The kernel image conforms to the Multiboot specification [5] used by QEMU [6] to set up the kernel for execution at boot time. This format is widely used by bootloaders [7]. CastorOS can thus be installed on real hardware using any Multiboot compatible bootloader. You do not need to run CastorOS on bare metal for the assignments.

To boot CastorOS in a QEMU virtual machine, run the following command from a terminal:

```
qemu-system-x86_64 \
-smp cpus=1 \
-kernel build/sys/castor \
-hda build/bootdisk.img \
-nic none \
-nographic
```

The backslashes (\) escape newlines, allowing the command to be split across multiple lines.

-smp cpus=1 tells QEMU to emulate a single CPU for now.

-kernel tells QEMU which kernel to boot.

-hda sets up an emulated IDE hard drive with the given image.

-nic none disables the Network Interface Card, and

-nographic disables the graphical framebuffer.

Instead, you will interact with the machine through a serial console directly in your terminal. The output should look something like this:

```
SeaBIOS (version 1.16.3-debian-1.16.3-2)
Booting from ROM..Castor Operating System
Invalid magic number: 0x0
flags = 0x24f
mem_lower = 639KB, mem_upper = 129920KB
boot_device = 0x8000ffff
cmdline = build/sys/castor
mods_count = 0, mods_addr = 0x43b000
mmap_addr = 0x9000, mmap_length = 0xa8
 size = 0x14, base_addr = 0x0, length = 0x9fc00, type = 0x1
 size = 0x14, base_addr = 0x9fc00, length = 0x400, type = 0x2
 size = 0x14, base_addr = 0xf0000, length = 0x10000, type = 0x2
 size = 0x14, base_addr = 0x100000, length = 0x7ee0000, type = 0x1
 size = 0x14, base_addr = 0x7fe0000, length = 0x20000, type = 0x2
 size = 0x14, base_addr = 0xfffc0000, length = 0x40000, type = 0x2
 size = 0x14, base_addr = 0xfd00000000, length = 0x300000000, type = 0x2
Initializing GDT... Done!
Initializing TSS... Done!
Initializing IDT... Done!
Initializing Syscall... Done!
Initializing PMAP ... Done!
Initializing XMEM ... Done!
AddRegion: 02000000 01f00000
RTC: Measuring CPU clock...
RTC: 1779211924 Ticks Per Second: 2399759121
LAPIC: CPU 0 found at 0xffff8000fee00900
IOAPIC: ID:0 Max Interrupts: 23
Booting on CPU 0
Starting processor 1
CR3: 0000000001fff000 RSP: ffff800003eb9000
Processor 1 did not respond in 250 ms
PCI: Initializing ...
PCI: (0,0,0) Host Bridge (8086:1237)
PCI: (0,1,0) ISA Bridge (8086:7000)
PCI: (0,1,0) ISA Bridge (8086:7000)
PCI: (0,1,1) IDE Controller (8086:7010)
PCI: (0,1,3) Other Bridge (8086:7113)
PCI: (0,2,0) VGA (1234:1111)
PCI: Initialization Done!
vfs: /sbin/init ffff810040003de0
loader:   Offset            VAddr FileSize  MemSize
loader: 00000000 0000000000400000 000001ec 000001ec
loader: AllocMap 0000000000400000 000001ec
loader: 00001000 0000000000401000 00004161 00004161
```

```
loader: AllocMap 0000000000401000 00004161
loader: 00006000 0000000000406000 000018bc 000018bc
loader: AllocMap 0000000000406000 000018bc
loader: 00008000 0000000000408000 00000200 000007b0
loader: AllocMap 0000000000408000 000007b0
loader: Jumping to userspace
CPU 0
Interrupt 14 Error Code: 0000000000000004
cr0: 0000000080050033 cr2: 0000000000000000
cr3: 0000000003ec5000 cr4: 00000000000006a0
dr0: 0000000000000000 dr1: 0000000000000000 dr2: 0000000000000000
dr3: 0000000000000000 dr6: 00000000ffff0ff0 dr7: 0000000000000400
rip: 0033:0000000000401010 rsp: 003b:000000007000f000
rflags: 0000000000000202 ds: 003b es: 0000 fs: 0000 gs: 0000
rax: 0000000000000000 rbx: 0000000000000000 rcx: 0000000000000000
rdx: 0000000000000000 rsi: 0000000000000000 rdi: 000000007000f000
rbp: 0000000000000000 r8:  0000000000000000 r9:  0000000000000000
r10: 0000000000000000 r11: 0000000000000000 r12: 0000000000000000
r13: 0000000000000000 r14: 0000000000000000 r15: 0000000000000000
Entered Debugger!
kdbg>
```

This means CastorOS started to boot up but ran into an error that you will fix in Assignment 1. Because there is an error, the debugger `kdbg` is automatically invoked. You can exit using the command:

```
ctrl-a x
```

3.4 Submitting Your Code

It takes four steps to submit your code. For example, if I had edited the file example.c in sys/dev I would do the following steps

1. Add any files you have edited into git. Only add (and commit) the files you are explicitly instructed to modify. In Assignment 0 it is one file and you have to figure out which one by the compile errors. In other assignments we will tell you the names of the files explicitly. Changes to other files will be rejected by the server. The exact path to the file will depend on where you are in the directory tree. For example,

   ```
   git add sys/dev/example.c
   ```

2. Commit those changes.

   ```
   git commit -m "A0 fix"
   ```

3. Move to the client directory (where `client.py` is located) and run the following command, which creates (or updates) a file called `castoros.patch`.

   ```
   python client.py patch
   ```

4. Finally submit the changes to the server using the following:

   ```
   python client.py submit -a asst0
   ```

This command sends the patch to the submission server and queues it for evaluation. The submission server typically takes about an hour to evaluate submissions. Submitting a new patch overwrites any active submissions without evaluating them.

Finally you can check your results using a command we saw before

```
python client.py status -a asst0
```

There are no secret or hidden tests. The mark you see is the mark you will receive on the assignment. If it is not perfect, you can submit again. We will always use the results from your most recent submission (not your best one) as the grade for the assignment.

If the code has the expected behaviour when you test it, but fails when you submit it, a possible cause is forgetting one of the required `git` or `patch` commands listed above. You can check that these have all been successfully completed by looking at the castoros.patch file. It should include all the changes you made to your code.

4 Setting Up on Windows 11 or macOS

For Windows 11, use the Windows Subsystem for Linux (WSL) [8].

For macOS, be aware that it does not use the ELF file format used by CastorOS. To build on a mac you will need to install Clang from your favorite package manager (assuming it includes lld support). Importantly the LINK parameter will point to your LLVM's ld.lld instead of the clang binary. For example it might look something like this depending where your clang distribution is installed:

```
CC="/opt/local/libexec/llvm-15/bin/clang"
LINK="/opt/local/libexec/llvm-15/bin/ld.lld"
```

References

[1] Amazon Web Services, *What is Virtualization?*, August 2024. Available at: `https://aws.amazon.com/what-is/virtualization`.

[2] *SCons: A software construction tool*, August 2024. Available at: `http://scons.org`

[3] *Clang: a C language family frontend for LLVM*, August 2024. Available at: `https://clang.llvm.org/`

[4] *The LLVM Compiler Infrastructure Project,* August 2024. Available at: `https://llvm.org`

[5] *Multiboot Specification*, August 2024. Available at: `https://www.gnu.org/software/grub/manual/multiboot/multiboot.html`

[6] *QEMU*, August 2024. Available at: `https://www.qemu.org/`

[7] *Bootloader*, August 2024. Available at: `https://en.wikipedia.org/wiki/Bootloader`

[8] *What is the Windows Subsystem for Linux?*, August 2024. Available at: `https://learn.microsoft.com/en-us/windows/wsl/about`