CS 423 Spring 2026 MP0: Setting Up a Kernel Development Environment

Due by 2/9 11:59 PM Central Time. No late submission.

This document will guide you through your MP0 for CS 423 Operating System Design. It will help you prepare an environment for upcoming MPs.

Considering that you need to compile a Linux kernel, this MP0 may consume a few hours for you to complete.

If you don't do this MP correctly, you will not be able to work on the later MPs.

Overview

Goals

- In this MP you will learn to download, compile, and test your own kernel.
- You will configure your development environment for upcoming MPs.
- The kernel source code will be a helpful reference.

Before You Start

What you need

Each student will need a Linux environment to complete all the MPs in CS 423. Specifically, you will need a Linux environment to compile your own kernel. We will then provide a QEMU script, and you will use QEMU to test your kernel and all the upcoming MPs.

As many of you do not own a Linux machine, we have requested a pool from Engineering IT., Every student will get an EngrIT VM running Linux from the university. You must use it for your MPs throughout the semester.

Before using the EngrIT VM, please review the policies at: https://answers.uillinois.edu/illinois.engineering/page.php?id=104597.

The EngrIT VM is preinstalled with Ubuntu, but it does not come with a GUI. You will need toconnect to your VM using SSH for development. Most modern operating systems ship with an SSH client, and you can use your favorite terminal emulator (like Windows Terminal) and development tools (like VSCode) to connect to your VM with campus VPN and your NetID.

You can view your VM assignment at: https://csid-basic-apps.cs.illinois.edu/. Please use campus VPN to access. If you cannot find your name, please let us (TAs) know. You can find our email addresses on the course home page. Please CC both of us.

Environmental Setup

EngrIT VM Setup

The EngrIT VM is already set up and ready for your use. The only two things for you to do is to power it up and connect to it using your favorite terminal emulator.

Please power on your VM:
https://answers.uillinois.edu/illinois.engineering/page.php?id=108475.

Development Tools Setup (Optional)

In this section, we introduce one recommended development tools setup. Note that this is not the only way to do it. You are free to use other setups for your MPs that you're comfortable with.

Public key

First, let's install your public key to the EngrIT VM so you don't have to type your password every time you connect to your VM. You can follow this guide: https://www.ssh.com/academy/ssh/copy-id.

VSCode

We recommend VSCode as your primary development tool. It is free on all platforms. VSCode is a powerful development tool that ships with a terminal emulator, a code editor, and a file browser, and it allows you to connect to your remote VM.

On the left bottom corner, you will find an icon that looks like "><". This is called "Remote Status" icon. Click it to see options to connect to your remote machine. Please follow this guide: https://code.visualstudio.com/blogs/2019/10/03/remote-ssh-tips-and-tricks, to add your EngrIT VM to your SSH config file. After this, you can conveniently use this small icon to connect to your EngrIT VM. If you have installed your public key correctly, VSCode shouldn't prompt for a password.

After successful connection, the Remote Status icon should show your VM's hostname. Then, you can navigate to the menu bar to open any folder, file, or bring up a terminal.

Kernel Compilation

Prepare for kernel compilation

If you followed the above development tools guide, you can now connect to your EngrIT VM. Otherwise, use your own favorite way to connect.

Configure your machine for kernel module development by downloading the required tool chains and QEMU for kernel development:

```
sudo apt-get update
```

```
sudo apt-get install git bc libncurses-dev wget busybox libssl-dev libelf-dev dwarves flex bison build-essential python3 qemu-system-x86
```

Feel free to install any packages and tools you find useful. You may find using the recommended VSCode is sufficient, or you can install your own editors you prefer, like Emacs and Vim.

Now, download the kernel source code:

```
wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.15.165.tar.xz
```

```
tar -xf linux-5.15.165.tar.xz
```

```
cd linux-5.15.165
```

You can check the version of your clone kernel source via the `make kernelversion` command. For CS 423, we will use kernel version 5.15.165.

Configure your kernel

You should clone the provided QEMU scripts and kernel config and the copy the config over:

```
git clone https://github.com/cs423-uiuc/qemu-script.git ../qemu-script
```

```
cp ../qemu-script/.config-x86-64 .config
```

To apply the kernel config, do:

```
make olddefconfig
```

This will create a config for your kernel based on the old .config file. In case there are any new config without values, it will use the default automatically.

Clean the kernel source directory to prepare for compilation:

```
make clean
```

Compile your kernel

We can now compile the kernel. Best practice when compiling the kernel is to parallelize the build, one thread per available core. This next command automatically detects the available CPUs on your VM to do this:

```
make -j`nproc` LOCALVERSION=-$NETID
```

Change the above line to replace `$NETID` with your NetID. `$` denotes environment variables in the shell and should also be removed. The `LOCALVERSION` field will be appended to the name of your kernel.

Upon successful compilation, you will have a new kernel image (a file called `vmlinux`) built in your kernel directory.

Test your kernel

You can now try to boot QEMU from your kernel source root directory with the following command:

```
../qemu-script/cs423-q
```

If you got permission denied when trying to start QEMU, make sure you have added yourself to the KVM group (`sudo usermod -a -G kvm your_username`). You can also use `sudo`.

If everything is correct, you should have a shell inside your VM. You can double check the kernel in the VM using the command below and should see your own kernel version (the one with your NetID):

```
uname -a
```

Example output:

```
Linux CSL-PowerEdge-R750 5.15.165-tyxu #1 SMP Sun Aug 13 04:57:03 EDT 2023
x86_64 x86_64 x86_64 GNU/Linux
```

Depending on your environment and configuration, your output may look different. But, please make sure the output contains `5.15.165-<netid>`.

To quit the QEMU and halt your kernel, you can simply exit the shell by typing exit. But just in case that you break your kernel, press Ctrl-A, and then press X. This will force QEMU to quit.

Finally, I have one tip about QEMU's filesystem. Using our provided QEMU script, your QEMU VM's root will be an overlayfs. You can access all the files just like in your host Linux, but all modifications are temporary. This means you need to recompile everytime you are using QEMU to test your kernel module unless you are compiling in the host Linux, and all edits you have done in QEMU will not be saved. However, it will sync the latest modifications from your host Linux, so each time you have edited the MP source file, you do not need to restart QEMU.

Navigate the Kernel Source

Bootlin Elixir

The easiest way to navigate through the kernel source is to use the Bootlin Elixir. It is available online and easy to use: https://elixir.bootlin.com/linux/v5.15.165/source. Make sure you choose the correct kernel version.

clangd

You can also use clangd with VSCode to navigate the kernel and MP source code. It can be particularly helpful for your development. Note that you are free to use other setups for your MPs that you're comfortable with.

clangd is a powerful language server that provides code completion, error highlighting, and refactoring. It is more accurate and much much faster for kernel development than the default Microsoft C/C++ extension.

Bring up a terminal, and install the necessary packages on the VM: `sudo apt install clangd bear`. Also, navigate to VSCode Extensions panel, search for clangd extension, and install it.

Installing clangd does not automatically make it usable. To make clangd work, we need to configure `compile_commands.json`. This file is project-dependent, i.e., you need to do this once for each MP. So you may want to refer back to this section before starting each MP.

To configure, open your MP folder in VSCode, and edit the Makefile. The provided Makefile only makes your MP compilable in our provided QEMU. We need to make your MP also compilable in your host Linux. We will include more information in the MP1 documentation.

After you edit Makefile, bring up a terminal and run `bear -- make` in your MP folder. Upon successful compilation, it will generate compile_commands.json for you, at which point clangd should be working. You can now enjoy powerful features like F12 jump, error highlighting, etc.

Submit Your Result

This is an easy one - you just need to submit to this form: https://forms.gle/Z6a9re5SVsumzCzR7 with the output from the following command:

```
# In your CS423 QEMU

dmesg | grep 'Linux version'
```

We kindly request that you login with your Illinois credentials to make sure the submitter is you. Please do not ask for permission.