Supplement for Kernel Module Development Environment

cfmc30

March 4, 2025

Outline

Presentation agenda

Why do we use QEMU?

- QEMU is an emulator that provides a controlled environment for testing your kernel.
  - Docker is a containerization technology, where processes inside the container share the host kernel.
    - This means that Docker is not suitable for kernel development since it does not provide kernel isolation.
  - If you want to learn more (optional), consider reading about:
    - Virtual Machine Hypervisors
    - Emulators
    - Containers
- Since we are developing a kernel module, which directly interacts with the kernel, using a virtual machine (VM) like QEMU helps isolate our development environment and prevents host system conflicts.

Environment to compile the kernel module?

- You can compile the kernel module using the `up-runtime` container.
  - If you want to run QEMU inside the container, you need to install `qemu-system-x86` in the `Dockerfile`.
    - A great opportunity to learn Docker, isn't it?
- For ARM64 macOS players, you need to build the kernel in a `crossbuild` environment.
  - You can also explore how to merge these two environments for a unified workflow.

Presentation agenda

Packing `rootfs`

- If you check `qemu.sh`, it launches QEMU and uses `rootfs.cpio.bz2` as the `initrd`, which acts as the root filesystem (`/`) in QEMU.
  - Your task is to:
    1. Extract `rootfs.cpio.bz2` into a directory named `rootfs`.
    2. Run `make install` to copy the compiled module into `rootfs/modules`.
    3. Archive and compress the directory back into `rootfs.cpio.bz2`.
  - Suggestions:
    - Automate these steps with a shell script to streamline development.

Presentation agenda

Major number and minor number

- You can check the major and minor numbers of device files using

  `$ ls -la /dev`

  Example output:

  `$ ls -la /dev/tty*`

```
crw-rw-rw- 1 root tty      5,  0 Mar  3 16:10 /dev/tty
crw--w---- 1 root tty      4,  0 Mar  1 07:27 /dev/tty0
crw--w---- 1 root tty      4,  1 Mar  1 07:28 /dev/tty1
crw--w---- 1 root tty      4, 10 Mar  1 07:27 /dev/tty10
```

  - The first number (e.g., 5 and 4) is the major number, which identifies the driver handling the device.
  - The second number (e.g., 0, 1, 10) is the minor number, which distinguishes multiple devices managed by the same driver.
  - To learn more, check
    The Linux Kernel Module Programming Guide #5.6 Device Drivers.

Presentation agenda

How to check if hellomod is inserted

- You can check `lsmod`, `/proc/modules`.
- If you check the code in `hellomod.c`, you will find the following line:

```
device_create(clazz, NULL, devnum, NULL, "hello_dev");
```

  - This function creates a device node in the `/dev` filesystem.
    - To verify, check if `/dev/hello_dev` exists.
  - Additionally, the code also creates `/proc/hello_mod` in a similar way.

How to interact with `hellomod`

- In `hello.c`:
  - This file provides examples demonstrating various ways to interact with `hellomod` using `read`, `write`, and `ioctl`.
  - You can execute the `hello` program to see the results.
- This topic is also covered in the course video: File + Stdio (+HW1).

Presentation agenda

Read the Examples

- Make sure to review the provided examples:
  - Code Example for Symmetric Key Cipher Operation
  - And others in the Pre-Lab Announcement
- Learn how to allocate memory in kernel space.
- Practice implementing read/write operations in a kernel module.

Experiment and Explore

- Have fun with kernel modules and the UNIX system!