Installing Kernel Development Tools on macOS

Alex Chi

Update: March 3, 2020

Contents

1 Goal of this tutorial

If you don't want to install VMware or other heavy virtual machine monitors for this course, you may go ahead and read this tutorial.

This tutorial will cover the following steps:

1. Compile the provided kernel on macOS
2. Run the kernel in supported emulator (In macOS or Docker)
3. Set up the development environment for Linux kernel module

2 Compile goldfish 3.4

2.1 Install Required Software

You still have to run a virtualization software called "Docker" on macOS to emulate a Linux-compatible environment. Otherwise, it would be hard for you to compile a kernel.

Required toolchains:

1. **Docker for macOS**
2. **Android NDK from course site**
   You may also download the latest version of Android NDK. However, as the latest version switched to use LLVM instead of GNU toolchains, and doesn't provide standalone gcc and g++ commands, it would be hard to compile the kernel.
3. **Android SDK from course site or from Android Developers**
   Our course provides a 10-year-ago SDK, which corresponds to Release 1.6 r1. As latest macOS dropped support for 32-bit applications, you won't be able to run the emulator on macOS. The latest version of SDK you may use is 24.x, where its emulator supports legacy qemu interface and kernel version below 3.10.
4. **goldfish kernel from course site or from AOSP**
   The kernel our course provides is Linux 3.4, with a config file allowing kernel module support inside. You may also clone goldfish 3.4 from AOSP or its mirror site in mirrors.tuna.tsinghua.edu.cn . You may have to run the config process yourself.
5. **Disk Utility on macOS**
   As macOS filesystem (HFS, APFS) is by default case-insensitive, it can't be used to store Linux kernel source code.
6. **VSCode with Remote Container plugin**
   It's a lot easier to set up Docker containers with VSCode.

2.2 Create a Case-sensitive Volume

In Disk Utility, select File - New Image, and create an APFS or HFS case-sensitive volume. Mount it. 10GB is enough for this course.

2.3 Decompress SDK, NDK and goldfish

In the mounted volume, decompress all these things.

2.4 Build kernel

Use VSCode to set up a container in your mounted volume. "Command + Shift + P" - Container - C++ (or other containers set up with build tools). By default this is a minimum Ubuntu installation.

After a few minutes, you may open integrated terminal in VSCode.

```
export NDK_HOME={YOUR NDK PATH}
export PATH="$NDK_HOME/toolchains/x86_64-4.9/prebuilt/linux-x86_64/bin":$PATH
export CROSS_COMPILE=arm-linux-androideabi-
export ARCH=armeabi
cd goldfish
make -j4
```

After that, you'll find zImage file in goldfish/arch/balabala/zImage

3 Run Kernel in Emulator

If you've downloaded r24 SDK, you may run this step in macOS. Otherwise, you have to run the following command in Docker. If so, you have to install jdk in container. Add `RUN apt install default-jdk -y` in `Dockerfile`.

```
./emulator @OsPrj -show-kernel -kernel balabala/zImage -no-window
```

After about 30 seconds, you'll be able to connect to emulator with adb.

```
adb devices
adb shell
```

You may test if kernel module works in shell by typing `lsmod`.

4 Set up Android Kernel Module Development Environment

From now on, you may follow steps provided in slides. Everything should work now. You may run NDK tools inside Docker container, and then push built kernel module.

After all, I'd highly recommend using a real Ubuntu virtual machine for this project. The reasons are that:

1. **Building kernel is much faster inside virtual machine.**
   As Docker has a overlay filesystem on host fs, there'll be huge performance issue while building goldfish. Building from AOSP source takes about 15 minutes.
2. **Prebuilt tools are already provided.**
   If you intend to run toolchains on macOS, you have to install latest compatible SDK (r24), and download Android system image from Google. For some of the students this process could be tough.
3. **VSCode provides easy access to VM.**
   Copy and paste things into and from virtual machine used to be a problem. However, after setting up SSH server in VM, you can easily access shell and files with VSCode Remote plugin, which is my current set up.