Embed Linux & Linux Device Driver Studies

The Four Elements of Embedded Linux

- Toolchain
- Bootloader
- Kernel
- Root File System

We can build these elements one by one manually, or we can also use build tools such as buildroot or yocto.

Linux Environment on Windows

- WSL
- Docker

WSL

Lists available distro for WSL from the internet..

```
wsl --list --online
```

Install distro.

```
wsl.exe --install [Distro]
```

Build & Install Host Linux Kernel

In WSL, if we want to build a kernel module, we need to build and install the linux kernel first.
Reference: [How to use the Microsoft Linux kernel v6 on Windows Subsystem for Linux version 2 (WSL2)]

```
git clone https://github.com/microsoft/WSL2-Linux-Kernel.git
sudo apt update && sudo apt install build-essential flex bison libssl-dev libelf-dev bc
python3 pahole cpio
cd WSL2-Linux-Kernel
sudo make -j$(nproc) KCONFIG_CONFIG=Microsoft/config-wsl
```

After make, we can find kernel image under <Linux_Source_Path>/arch/x86/boot/bzImage

```
  CC      arch/x86/boot/compressed/mem.o
  CC      arch/x86/boot/compressed/efi.o
  CPUSTR  arch/x86/boot/cpustr.h
  CC      arch/x86/boot/cpu.o
  CC      arch/x86/boot/compressed/misc.o
  GZIP    arch/x86/boot/compressed/vmlinux.bin.gz
  MKPIGGY arch/x86/boot/compressed/piggy.S
  AS      arch/x86/boot/compressed/piggy.o
  LD      arch/x86/boot/compressed/vmlinux
  ZOFFSET arch/x86/boot/zoffset.h
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#2)
root@10d0e0995485:/workspace/WSL2-Linux-Kernel#
```

Then install the built modules and headers to /lib/modules/<uname -r>.

```
sudo make modules_install headers_install
```

Install kmod

There's no insmod and rmmod by default in WSL.
We need to install kmod by ourselves.

```
sudo apt-get install kmod
```

Installing New Microsoft Linux Kernel

1. Create or edit the file %USERPROFILE%\.wslconfig with the following content:

```
[wsl2]
kernel="C:\\<IMAGE_PATH>\\bzImage"
```

2. Restart WSL

Docker

Docker Desktop Setup on Windows

(1) Install Docker Desktop Installer.exe
(2) Pull Ubuntu image from docker hub

```
docker pull ubuntu:22.04
```

(3) Check images in local docker

```
docker image ls
```

(4) Create a container

```
docker run -dti --volume <LOCAL_WORK_PATH>:<CONTAINER_WORK_PATH> --workdir <WORK_DIR> --rm
--name <CONTAINER_NAME> <IMAGE-ID> tail -f /dev/null
```

e.g.

```
# Create container in WSL
docker run -dti --volume /mnt/c/workspace:/workspace --workdir /workspace --rm --name
ubuntu-22.04 1ec65b271951 tail -f /dev/null

# Create container in DOS
docker run -dti --volume C:\workspace:/workspace --workdir /workspace --rm --name ubuntu-22.04
1ec65b271951 tail -f /dev/null
```

(5) Check container

```
docker container ls
```

(6) Run container

```
docker exec -it <CONTAINER_NAME> /bin/bash
```

e.g.

```
docker exec -it ubuntu-22.04 /bin/bash
```

Login as user rather than root

```
docker exec -it --user <USER_NAME> <CONTAINER_NAME> /bin/bash
```

e.g.

```
docker exec -it --user muheng ubuntu-22.04 /bin/bash
```

To run a command as administrator, need to install sudo

```
apt-get install sudo
```

Unable to execute mount and dmesg

Create the container with privileged enabled.

```
--privileged=true
```

e.g.

```
# Create container in DOS
docker run -dti --volume C:\workspace:/workspace --workdir /workspace --rm --name ubuntu-22.04
--privileged=true 1ec65b271951 tail -f /dev/null
```

Embedded Linux Emulator

Install QEMU in Ubuntu 22.04

(1) Update ubuntu

```
apt-get update
```

(2) Search Qemu

```
apt search qemu*
```

(3) Install qemu-system-arm

```
apt install qemu-system-arm
```

(4) Check supported machine type

```
qemu-system-arm -machine help
```

Linux Setup

Install build essential

From Udemy LDD course

```
apt-get install build-essential lzop u-boot-tools net-tools bison flex libssl-dev
libncurses5-dev libncursesw5-dev unzip chrpath xz-utils minicom
```

From my study

```
apt-get install gcc build-essential automake gcc-arm-linux-gnueabihf vim git wget python3
pkg-config zlib1g-dev libglib2.0-dev libpixman-1-dev flex bison unzip libncurses5-dev
```

Required by using docker image Ubuntu 22.04

```
apt-get install vim file wget cpio rsync bc
```

Create A New User

(1) Create a user with home directory

```
useradd -m <user_name>
```

(2) Set password

```
passwd <user_name>
```

(3) Add the new user to sudo group

```
usermod -aG sudo <user_name>
```

(4) Confirm user created and details

```
id <user_name>
```

BuildRoot

Custom Make

```
make menuconfig
```

Default Make

(1) Show default configurations

```
make list-defconfigs
```

(2) make config

```
make qemu_arm_vexpress_defconfig
```

(3) make

```
make
```

(4) The build target will locate in output/images
rootfs.ext2: root filesystem
zImage: kernel image
vexpress-v2p-ca9.dtb: device tree file

```
Creating regular file /workspace/buildroot/output/images/rootfs.ext2
Creating filesystem with 65536 1k blocks and 16384 inodes
Filesystem UUID: d0f29b93-a25a-4b5a-ab3a-e024224475ad
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345

Allocating group tables: done
Writing inode tables: done
Copying files into the device: done
Writing superblocks and filesystem accounting information: done

ln -snf /workspace/buildroot/output/host/arm-buildroot-linux-gnueabihf/sysroot /workspace/buildroot/output/staging
>>>   Executing post-image script board/qemu/post-image.sh
muheng@f25456561417:/workspace/buildroot$ ls output/images/
rootfs.ext2  start-qemu.sh  vexpress-v2p-ca9.dtb  zImage
muheng@f25456561417:/workspace/buildroot$
muheng@f25456561417:/workspace/buildroot$
muheng@f25456561417:/workspace/buildroot$
```

Trouble Shooting

(1) Should Not Build As Root

```
checking for mknodat... yes
checking for mkfifoat... yes
checking whether mkfifoat rejects trailing slashes... yes
checking whether mknod can create fifo without root privileges... configure: error: in `/workspace/buildroot-2025.05.1/output/build/host-tar-1.35':
configure: error: you should not run configure as root (set FORCE_UNSAFE_CONFIGURE=1 in environment to bypass this check)
See `config.log' for more details
make: *** [package/pkg-generic.mk:263: /workspace/buildroot-2025.05.1/output/build/host-tar-1.35/.stamp_configured] Error 1
root@67b85db06a6a:/workspace/buildroot-2025.05.1#
```

Solution: Create A New User

(2) Login As User To Docker Container

```
docker exec -it --user <USER_NAME> <CONTAINER_NAME> /bin/bash
```

e.g.

```
docker exec -it --user muheng ubuntu-22.04 /bin/bash
```

(3) Buildroot Build Errors

[1] Build on WSL

While trying to build on WSL, will get below error:

```
Your PATH contains spaces, TABs, and/or newline (\n) characters.
This doesn't work. Fix you PATH.
make: *** [support/dependencies/dependencies.mk:27: dependencies] Error 1
```


To resolve this error, we can add following script into “~/.bashrc”.

```
vi ~/.bashrc
```

Script to append to the end of “~/.bashrc”:

```
PATH=$(/usr/bin/printenv PATH | /usr/bin/perl -ne 'print join(":", grep { !/\/mnt\/[a-z]/ } split(/:/));')
```

[2] Failed to build libffi

Error log:

```
configure: WARNING: unrecognized options: --disable-gtk-doc, --disable-gtk-doc-html, --disable-doc, --disable-documentation, --with-xmlto, --with-fop, --disable-nls
../../configure: line 2202: config.log: No such file or directory
../../configure: line 2212: config.log: No such file or directory
cat: standard output: No such file or directory
make: *** [package/pkg-generic.mk:263: /workspace/buildroot/output/build/host-libffi-3.4.8/.stamp_configured] Error 1
```

Resolution:

```
vi output/build/host-libffi-3.4.8/m4/ax_enable_builddir.m4
```

Goto line 119, find below code

```
test -f $srcdir/config.log   && mv $srcdir/config.log   .
```

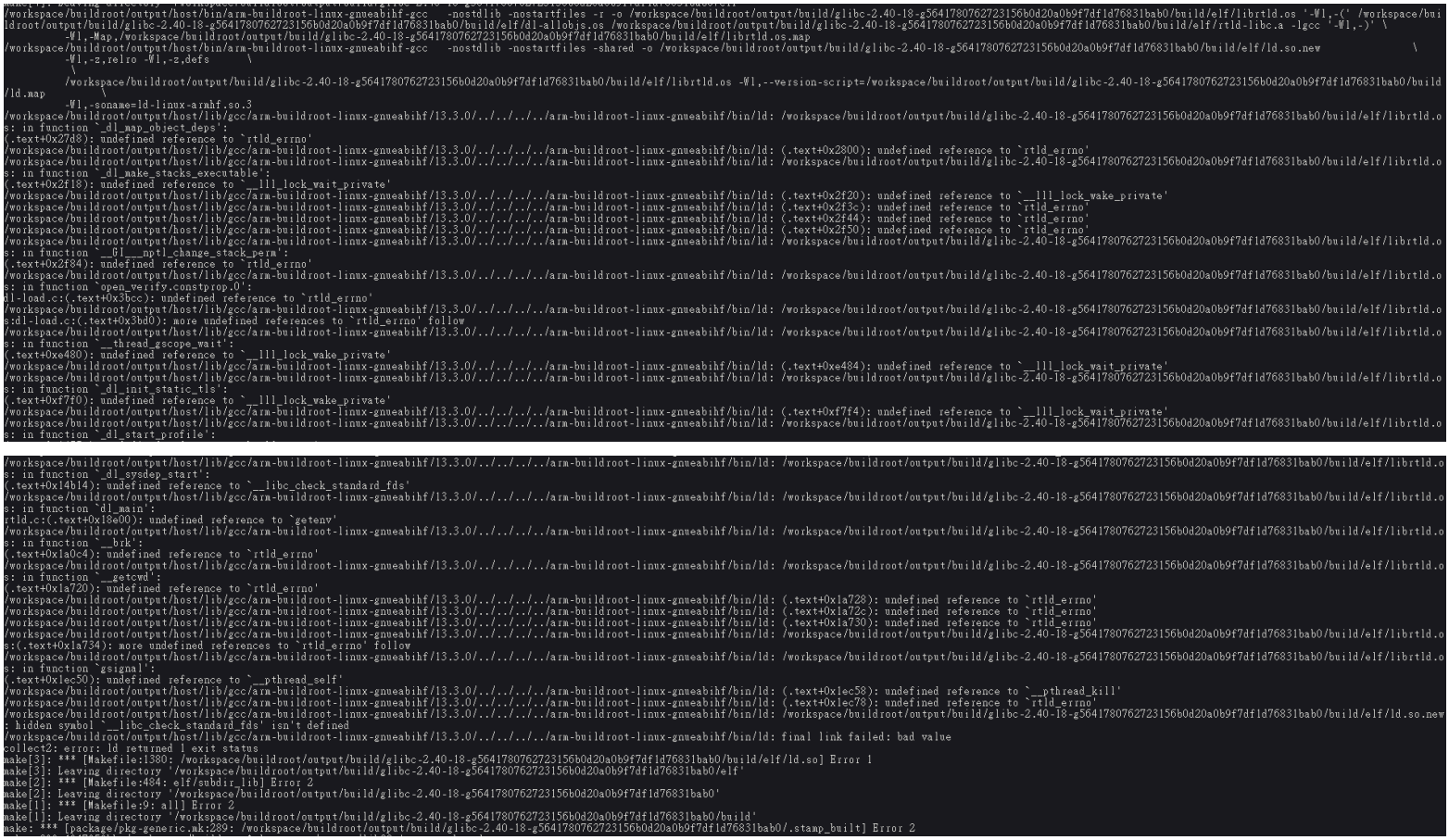
Replace mv with cp

```
test -f $srcdir/config.log   && cp $srcdir/config.log   .
```

Reference Link:

[WSL编译buildroot相关问题解决]

[3] Failed to build glibc, undefined reference to glibc-2.51-5-xxx/elf/librtld.os



The error was caused by Windows case sensitive settings.
To avoid the build error, the case sensitive setting **MUST** be **enabled**.

To query the case sensitive setting of build path:

```
fsutil.exe file queryCaseSensitiveInfo <DIR_PATH>
```

e.g.

```
fsutil.exe file queryCaseSensitiveInfo C:\workspace\buildroot
```

If not enabled, use the below command to enable it.

```
fsutil.exe file setCaseSensitiveInfo <DIR_PATH> enable
```

e.g.

```
fsutil.exe file setCaseSensitiveInfo C:\workspace\buildroot enable
```

Reference Link:
WSL子系统编译buildroot填坑

Copy module to root filesystem image

```
mkdir tmpfs
sudo mount -o loop <buildroot_path>/output/images/rootfs.ext2 tmpfs/
sudo cp /workspace/ldd_study/empty_module/empty_module.ko tmpfs/lib/modules
sudo umount tmpfs
```

Kernel Module

Create Module Folder

```
mkdir <WORK_DIR>/empty_module
```

e.g.

```
mkdir /workspace/ldd_study/empty_module
```

Empty Module Example

Edit source

```
vi <WORK_DIR>/empty_module/empty_module.c
```

Empty module source code

```
#include <linux/module.h>

/**
 * module clean-up entry point
 * Returns 0 to indicate module init successfully,
 * otherwise, the module will not be loaded.
 */
static int empty_module_init(void) {
    pr_info("Init Empty Module...");
    return 0;
}

/** module clean-up entry point */
static void empty_module_exit(void) {
    pr_info("Exit Empty Module...");
}

/* register module entry point to kernel */
module_init(empty_module_init);
/* register module clean-up entry point to kernel */
module_exit(empty_module_exit);

MODULE_DESCRIPTION("Module Getting Started");
MODULE_VERSION("1.0");
MODULE_AUTHOR("Muheng Lee");
MODULE_LICENSE("GPL");
```

Build Module For Target Machine

Makefile

Edit Makefile

```
vi <WORK_DIR>/empty_module/Makefile
```

Makefile content

```
PWD := $(shell pwd)
KERNEL_DIR = /workspace/buildroot/output/build/linux-6.12.27

MODULE_NAME = empty_module
# obj-y: build static linked module
# obj-n: not build
# obj-m: build dynamic linked module
obj-m := $(MODULE_NAME).o

all:
	make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf- \
		-C $(KERNEL_DIR) M=$(PWD) modules
clean:
	make -C $(KERNEL_DIR) M=$(PWD) clean
```

Build Steps

Go to module directory

```
cd <WORK_DIR>/empty_module
```

Set tool-chain path

```
export PATH=$PATH:/workspace/buildroot/output/host/bin
```

Build

```
make
```

Reuslt

```
muheng@da2652a14654:/workspace/my_modules/empty_module$ make -j2
make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf- \
        -C /workspace/buildroot/output/build/linux-6.12.27 M=/workspace/my_modules/empty_module modules
make[1]: warning: jobserver unavailable: using -j1.  Add '+' to parent make rule.
make[1]: Entering directory '/workspace/buildroot/output/build/linux-6.12.27'
  CC [M]  /workspace/my_modules/empty_module/empty_module.o
  MODPOST /workspace/my_modules/empty_module/Module.symvers
  CC [M]  /workspace/my_modules/empty_module/empty_module.mod.o
  CC [M]  /workspace/my_modules/empty_module/.module-common.o
  LD [M]  /workspace/my_modules/empty_module/empty_module.ko
make[1]: Leaving directory '/workspace/buildroot/output/build/linux-6.12.27'
muheng@da2652a14654:/workspace/my_modules/empty_module$ ll
total 200
drwxr-xr-x 1 muheng muheng  4096 Aug 21 15:42 ./
drwxr-xr-x 1 muheng muheng  4096 Aug 21 15:00 ../
-rw-r--r-- 1 muheng muheng 34408 Aug 21 15:42 ..module-common.o.cmd
-rw-r--r-- 1 muheng muheng   220 Aug 21 15:42 .Module.symvers.cmd
-rw-r--r-- 1 muheng muheng   390 Aug 21 15:42 .empty_module.ko.cmd
-rw-r--r-- 1 muheng muheng   219 Aug 21 15:42 .empty_module.mod.cmd
-rw-r--r-- 1 muheng muheng 34189 Aug 21 15:42 .empty_module.mod.o.cmd
-rw-r--r-- 1 muheng muheng 34074 Aug 21 15:42 .empty_module.o.cmd
-rw-r--r-- 1 muheng muheng  6272 Aug 21 15:42 .module-common.o
-rw-r--r-- 1 muheng muheng   177 Aug 21 15:42 .modules.order.cmd
-rwxrwxrwx 1 root   root     388 Aug 21 14:55 Makefile*
-rw-r--r-- 1 muheng muheng     0 Aug 21 15:42 Module.symvers
-rwxrwxrwx 1 root   root     648 Aug 21 03:01 empty_module.c*
-rw-r--r-- 1 muheng muheng 39920 Aug 21 15:42 empty_module.ko
-rw-r--r-- 1 muheng muheng    50 Aug 21 15:42 empty_module.mod
-rw-r--r-- 1 muheng muheng   426 Aug 21 15:42 empty_module.mod.c
-rw-r--r-- 1 muheng muheng 27540 Aug 21 15:42 empty_module.mod.o
-rw-r--r-- 1 muheng muheng  8220 Aug 21 15:42 empty_module.o
-rw-r--r-- 1 muheng muheng    50 Aug 21 15:42 modules.order
muheng@da2652a14654:/workspace/my_modules/empty_module$
```

Build Module For Host Machine

Makefile

```
PWD := $(shell pwd)
KERNEL_DIR = /lib/modules/6.6.87.2-microsoft-standard-WSL2+/build

MODULE_NAME = empty_module
# obj-y: build static linked module
# obj-n: not build
# obj-m: build dynamic linked module
obj-m := $(MODULE_NAME).o

all:
	make -C $(KERNEL_DIR) M=$(PWD) modules
clean:
	make -C $(KERNEL_DIR) M=$(PWD) clean
```

Load / Unload Module

insmod <MODULE_NAME>.ko
rmmod <MODULE_NAME>

modprobe <MODULE_NAME>
modprobe -r <MODULE_NAME>

Linux Device Driver

Device Types

- Character Devices
- Block Devices
- Network Devices

Device Number (32bits)

<major_num>:<minor_num>
major_num (12 bits): indicating the driver being used to access the device file.
minor_num (20 bits): indicating which device file is being accessed.
The device number is defined as dev_t in the kernel.
There are two macros to help us extract the major and minor numbers from dev_t.

```
/* linux-headers-6.12.27/include/linux/types.h */
typedef u32 __kernel_dev_t;
typedef __kernel_dev_t dev_t;

/* linux-headers-6.12.27/include/linux/kdev_t.h */
#define MAJOR(dev)    ((unsigned int) ((dev) >> MINORBITS))
#define MINOR(dev)    ((unsigned int) ((dev) & MINORMASK))
```

We can use these macros like below:

```
dev_t device_num;
unsigned int major_num = MAJOR(device_num);
unsigned int minor_num = MINOR(device_num);
```

There is a macro that helps to convert major number and minor number to dev_t.

```
/* linux-headers-6.12.27/include/linux/kdev_t.h */
#define MKDEV(ma,mi)    (((ma) << MINORBITS) | (mi))
```

e.g.

```
dev_t dev_num = MKDEV(major_num, minor_num);
```

Program

Creation

alloc_chrdev_region	Dynamically allocate a device number to the char device.	linux/fs.h

Removal

unregister_chrdev_region	linux/fs.h

Device File

User applications exchange data with the driver through a device file.
We can manually create a device file by mknod or dynamically create a device file in the driver program.

mknod

```
mknod -m <MODE> /dev/<name> <DEV_TYPE> <MAJOR_NUM> <MINOR_NUM>
```

DEV_TYPE could be c as character device, b as block device, or p as pipe.
e.g.

```
mknod -m 666 /dev/hank0 c 248 0
```

Program

Creation

class_create	Create a new class under /sys/class. e.g. /sys/class/empty_char_device **Note.** **Starting from kernel v6.4+, the owner argument was removed.** **You will get the compiler error like below if you pass THIS_MODULE as the 1st argument:** **“error: passing argument 1 of ‘class_create’ from incompatible pointer type”**	linux/device.h
device_create	(1) Create subdirectories and uevent of all devices belonging to this class under /sys/class/<CLASS_NAME>/<DEVICE#>. e.g. /sys/class/empty_char_device/emp0 /sys/class/empty_char_device/emp1 /sys/class/empty_char_device/emp2 (2) The uevent triggers udev to create device files under /dev. /dev/emp0 /dev/emp1 /dev/emp2	linux/device.h

After device creation, the device file will be found under /dev. e.g. /dev/emp0.

Removal

device_destroy	linux/device.h
class_destroy	linux/device.h

Empty Char Driver Example

```c
#include <linux/cdev.h>
#include <linux/export.h>
#include <linux/fs.h>
#include <linux/module.h>

#define DEVICE_NAME "empty_char_device"
#define NUM_DEVICES 1

static dev_t dev_num = 0;
static struct cdev empty_cdev;
static struct file_operations empty_cdev_fops = {
    .owner = THIS_MODULE
};
static struct class *cls;
```

```c
/**
 * Module initialization entry point
 */
static int empty_char_driver_init(void) {
    pr_info("[empty_char_driver_init] Init Empty Char Driver...\n");
    /* Request kernel to allocates a range of char device numbers,
       the major number will be assgined dynamically. */
    int ret = alloc_chrdev_region(&dev_num, 0, NUM_DEVICES, DEVICE_NAME);
    if (ret != 0) {
        pr_err("[empty_char_driver_init] Failed to init Empty Char Driver...\n");
        return ret;
    }
    pr_info("[empty_char_driver_init] Create device number: <%d:%d>\n", MAJOR(dev_num),
MINOR(dev_num));

    /* Iniitialize cdev variable */
    cdev_init(&empty_cdev, &empty_cdev_fops);
    empty_cdev.owner = THIS_MODULE;

    /* Register the char device to VFS */
    cdev_add(&empty_cdev, dev_num, NUM_DEVICES);

    /* Starting from kernel v6.4+, the owner argument was removed. */
    cls = class_create(/*THIS_MODULE, */DEVICE_NAME);

    device_create(cls, NULL, MKDEV(MAJOR(dev_num), 0), NULL, "ecd");

    pr_info("[empty_char_driver_init] Empty Char Driver Loaded...\n");

    return 0;
}

/** module clean-up entry point */
static void empty_module_exit(void) {
    pr_info("[empty_module_exit] Exit Empty Char Driver...\n");
    device_destroy(cls, dev_num);
    class_destroy(cls);
    cdev_del(&empty_cdev);
    /* Release the device numbers */
    unregister_chrdev_region(dev_num, NUM_DEVICES);
    pr_info("[empty_module_exit] Empty Char Driver Unloaded...\n");
}

/* Register module entry point to kernel */
module_init(empty_char_driver_init);
/* Register module clean-up entry point to kernel */
module_exit(empty_module_exit);

MODULE_DESCRIPTION("Linux Device Driver Getting Started");
```

```
MODULE_VERSION("1.0");
MODULE_AUTHOR("Muheng Lee");
MODULE_LICENSE("GPL");
```

After inserting the module, we can find the driver class under /sys/class.

```
muheng@LAPTOP-N9PBJ2PF:/mnt/c/workspace/ldd_study/empty_char_driver$ ll /sys/class/empty_char_device/ecd/
total 0
drwxr-xr-x 3 root root    0 Sep  2 22:42 ./
drwxr-xr-x 3 root root    0 Sep  2 22:42 ../
-r--r--r-- 1 root root 4096 Sep  2 22:43 dev
drwxr-xr-x 2 root root    0 Sep  2 22:43 power/
lrwxrwxrwx 1 root root    0 Sep  2 22:43 subsystem -> ../../../../class/empty_char_device/
-rw-r--r-- 1 root root 4096 Sep  2 22:42 uevent
muheng@LAPTOP-N9PBJ2PF:/mnt/c/workspace/ldd_study/empty_char_driver$ cat /sys/class/empty_char_device/ecd/dev
240:0
muheng@LAPTOP-N9PBJ2PF:/mnt/c/workspace/ldd_study/empty_char_driver$ cat /sys/class/empty_char_device/ecd/uevent
MAJOR=240
MINOR=0
DEVNAME=ecd
```

And we can find the device file created by the driver.

```
muheng@LAPTOP-N9PBJ2PF:/mnt/c/workspace/ldd_study/empty_char_driver$ ll /dev/ecd
crw------- 1 root root 240, 0 Sep  2 22:42 /dev/ecd
```

File Operations Implementation

Multiple Device Nodes

References

[How to use the Microsoft Linux kernel v6 on Windows Subsystem for Linux version 2 (WSL2)]
[WSL编译buildroot相关问题解决]
[WSL子系统编译buildroot填坑]

Other Useful Links

[Online Linux Kernel Source Code]
[buildRoot study - 建立自己的作業系統]
[核心模組的載入與移除：insmod, modprobe, rmmod]
[mknod 與 device driver]
[WSL2连接USB存储设备]