Embedded Book Club

# Linux Device Driver Development
# Chapter 2: Understanding Linux Kernel Module Basic Concepts

**Jonathan Velasco**

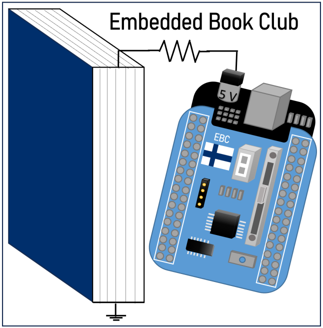
September 18th, 2023

# Embedded Book Club Finland

We're knowledge sharing enthusiasts, focused on hosting in-person events, to bond over technical topics related to embedded systems.

Our goal is to

- Create a community focused on Embedded Systems and related topics
- Share knowledge, and to learn about new topics, trends and practices
- Having fun learning and working on projects together



# Chapter 2: Understanding Linux Kernel Module Basic Concepts

**This chapter covers**

- Introduction to the concept of modules,
- Building a kernel module
- Exporting symbols and module dependencies

## Chapter 2: Understanding Linux Kernel Module Basic Concepts
## Kernel Modules

When building the Kernel, a single file (image) is created with all the corresponding features. One of these features is the support for loading/unloading of modules

- CONFIG_MODULES=y,
- CONFIG_MODULE_UNLOAD=y
- CONFIG_FORCE_UNLOAD=y

Modules can be static modules (built-in) or dynamically as a kernel loadable module.
Some examples of features that can be compiled as loadable modules are device drivers, filesystems and frameworks.

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Kernel Modules: Hello World

```
#include <linux/module.h>        /* Needed by all modules */
#include <linux/kernel.h>        /* Needed for KERN_INFO */

static int __init init_module(void)
{
        printk(KERN_INFO "Hello␣world␣!.\n");
    /*
        * A non 0 return means init_module failed; module can't be loaded.
        */
    return 0;
}

static void __exit exit_module(void)
{
    printk(KERN_INFO "Goodbye␣world␣!.\n");
}


module_init(init_module);
module_exit(exit_module);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Embedded␣Club␣Finland");
```

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Kernel Modules: __init and __exit attributes

The __init and __exit are kernel macros that tell the linker to place the symbols prefix in a dedicated section in the resulting kernel object file

- # define __init __section(.init.text)
- # define __exit __section(.exit.text)

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Kernel Modules: Information and Metadata

The module should have information about itself. Particular attention must be paid to the License type and symbols exported

- MODULE_LICENSE("GPL");
  - If License is Propietary, it will taint the kernel. It has an effect on module behavior, as it's not being able to see/use symbols exported by the kernel
- MODULE_EXPORT_SYMBOL() and MODULE_EXPORT_SYMBOL_GPL()
  - The GPL variant shows only GPL-compatible modules

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Building a Linux Kernel Module

- A module can be built statically as part of the Kernel tree or as a loadable kernel module (LKM)
- The linux kernel maintains its own build system - kbuild. There are three files that are part of this: Kconfig (feature selection), Kbuild and Makefile (for compilation rules)
- The dedicated tool to parse makefiles is called **make**

```
make -C $KERNEL_SRC M=$(shell pwd) [target]
```

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Makefile example/Out-of-tree

Using the ARCH and CROSS_COMPILE variables to set the right cross compiler for the particular CPU, in addition to the Makefile sample below, enable the user to do out-of-tree module cross compiling, which produces .ko objects that can be used as LKM

```
obj-m := hello.o

KERNEL_SRC ?= /lib/modules/$(shell uname -r)/build

all default: modules
modules help clean:
make -C $KERNEL_SRC M=$(shell pwd) @
```

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - In-tree build

This requires extra modifications in the build system files.

- Add definition of your module/driver in Kconfig
  - config EBC_DEV
    tristate ”Embedded Club Finland Character Driver”
    default m
    help
    Select Y to enable support
- Modify Makefile to account for enable/disable options
  - obj-$(CONFIG_EBC_DEV) += hello.o
- To build as loadable kernel module, add to your defconfig (under arch/)
  - CONFIG_EBC_DEV=m

This is what embedded board manufacturers do in order to provide Board Suport Packages (BSP). If you use menuconfig, you should see ”Embedded Club Finland Character Driver” as an option

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Handling Module Parameters

Similar to stdin, the kernel modules can take in arguments dynamically. You can expose variables to the terminal to adapt the behavior of the module according to parameters.

- module_param(name, type, perm);
- module_param_array(name, type, perm);

name is the name of the variable, type the variable type, and perm the file permissions (e.g., S_IUSR, S_IRUGO)
These macros are defined in *include/linux/moduleparam.h*

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Loading/Unloading Kernel modules

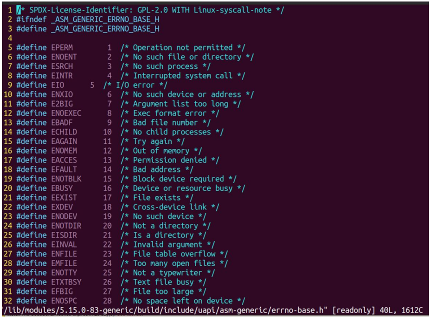
Utilities used in terminal

- depmod: Generates module dependency files and processes module files in order to extract and gather devices supported by the driver. The mapping is generated under modules.alias
- insmod: manual loading of kernel module module. Preferred choice during development
- modprobe: loading kernel module, preferred choice in production systems. It parses dependencies first, in order to load dependencies first.
- rmmod: unload kernel module

You can also load modules automatically at boot time by adding modules line under /etc/modules-load.d/filename.conf. Where filename.conf is your own configuration file.

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Error Handling

Return the right error code. To keep things neat, check-out the different error numbers under *include/uapi/asm-generic/errno-base.h*. The use of goto is also encouraged



```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
#ifndef _ASM_GENERIC_ERRNO_BASE_H
#define _ASM_GENERIC_ERRNO_BASE_H

#define EPERM        1  /* Operation not permitted */
#define ENOENT       2  /* No such file or directory */
#define ESRCH        3  /* No such process */
#define EINTR        4  /* Interrupted system call */
#define EIO      5  /* I/O error */
#define ENXIO        6  /* No such device or address */
#define E2BIG        7  /* Argument list too long */
#define ENOEXEC      8  /* Exec format error */
#define EBADF        9  /* Bad file number */
#define ECHILD      10  /* No child processes */
#define EAGAIN      11  /* Try again */
#define ENOMEM      12  /* Out of memory */
#define EACCES      13  /* Permission denied */
#define EFAULT      14  /* Bad address */
#define ENOTBLK     15  /* Block device required */
#define EBUSY       16  /* Device or resource busy */
#define EEXIST      17  /* File exists */
#define EXDEV       18  /* Cross-device link */
#define ENODEV      19  /* No such device */
#define ENOTDIR     20  /* Not a directory */
#define EISDIR      21  /* Is a directory */
#define EINVAL      22  /* Invalid argument */
#define ENFILE      23  /* File table overflow */
#define EMFILE      24  /* Too many open files */
#define ENOTTY      25  /* Not a typewriter */
#define ETXTBSY     26  /* Text file busy */
#define EFBIG       27  /* File too large */
#define ENOSPC      28  /* No space left on device */
"/lib/modules/5.15.0-83-generic/build/include/uapi/asm-generic/errno-base.h" [readonly] 40L, 1612C
```

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - Null pointer error handling

When you're returning from a function, it is possible that the function will return a NULL pointe. For that purpose the kernel provides the following functions

- void *ERR_PTR(long error); - Error value to pointer macro
- long IS_ERR(const void *ptr); - Check whether value is a pointer error
- long PTR_ERR(const void *ptr); - Pointer to error code

# Chapter 2: Understanding Linux Kernel Module Basic Concepts - printk

printk is the first debugging technique. It behaves similarly to printf in C. printk is the low-level printing API, however it is recommended using the following wrappers in new drivers

- pr_level
- dev_level(struct device *dev ...)
- netdev_level(struc net_device *dev

You can also customize the print messages

- #define pr_fmt(fmt) ''

```
$ cat /proc/sys/kernel/printk
```

Values: current log level, default log level, lowest console level and highest.