



Linux Kernel Modules

Changes in kernel 2.6

Lecturer:

Roberto Farina

CEFRIEL

roberto.farina@cefriel.it

Overview



- What's new in 2.6
- Updating the basic structure of a module
- Changes in the build process for LKMs
- Other considerations about 2.6 kernels

What's new in 2.6



- **Module versioning support**
 - Marked as **EXPERIMENTAL**
 - It adds **extra** versioning information to compiled modules at build-time
 - Designed to help increase module **portability** to kernels other than the one that they were compiled against
- **Module unloading option**
 - It must be enabled if you want your kernel to be able to unload modules when they are **no longer needed**
 - Especially important in **resource-constrained** and power-sensitive environment such as **embedded systems**
 - **Forced module unloading** to be able to forcibly unload modules even if the kernel believes they are in use

Device drivers



- 2.6 kernels introduce a **new unified framework** for device drivers
 - It requires changes to custom device drivers that you may have developed to run under earlier versions of the Linux kernel
 - **Full** and **complete** support for Plug and Play and Power Management
 - It defines the **interfaces** that subsystems can use when **communicating** with individual drivers
- **Sysfs** filesystem to provide a **hierarchical** view of each system's device tree

Updating the basic structure



- You must use the `module_init()` and `module_exit()` macros to register the names of your initialization and exit routines
 - They are only strictly necessary if you intend to compile the specified module into the kernel
- `#define MODULE` statement is no longer necessary
 - Automatically defined and verified by the kernel build system
- Use of the `MODULE_LICENSE` macro is strongly recommended

Changes to the build process



- Integration of external module compilation into the standard kernel build mechanism
 - You don't have to manually specify module-oriented declarations such as **MODULE**, **__KERNEL__**
 - You don't need to specify the optimization flag
- New naming convention
 - **.ko** (kernel objects) instead of **.o**
 - It is necessary to modify initialization scripts in order to take into account the new naming convention

Makefile and compilation



➤ Makefile

```
obj-m := minmod.o
```

➤ Compilation

- A simple command line to be executed from within the directory containing your module's source code

```
make -C /usr/src/linux SUBDIRS=$PWD modules
```

➤ The compilation step produces the basic object file

➤ The linker/loader step links in the file **init/vermagic.o**

- It **integrates information** about the kernel against which the module was compiled and provides more **stringent checks** used when loading the module

Other changes in 2.6



- New interface used for modules that take parameters
 - The `MODULE_PARM()` macro has been replaced by explicit parameter declarations made using the new `module_param()` macro
 - `moduleparam.h`
- Enhanced preemptability and SMP-awareness introduce some **new concerns** for driver writers
 - Drivers should use a spinlock or mutex to **protect** data that could be accessed from multiple processors
- Module reference counts are managed and manipulated differently