# Kernel Module

Chung-Ang University

# What is Kernel Modules in Linux

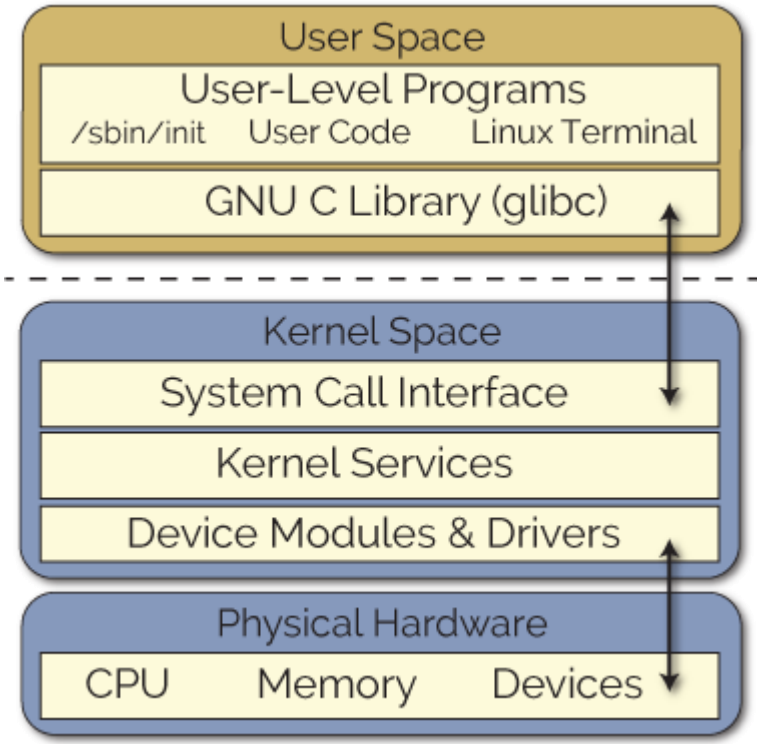
- Modules are pieces of code that can be loaded and unloaded into the kernel upon demand
- They extend the functionality of the kernel without the need to reboot the system
  - For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system
  - Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image
  - Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality

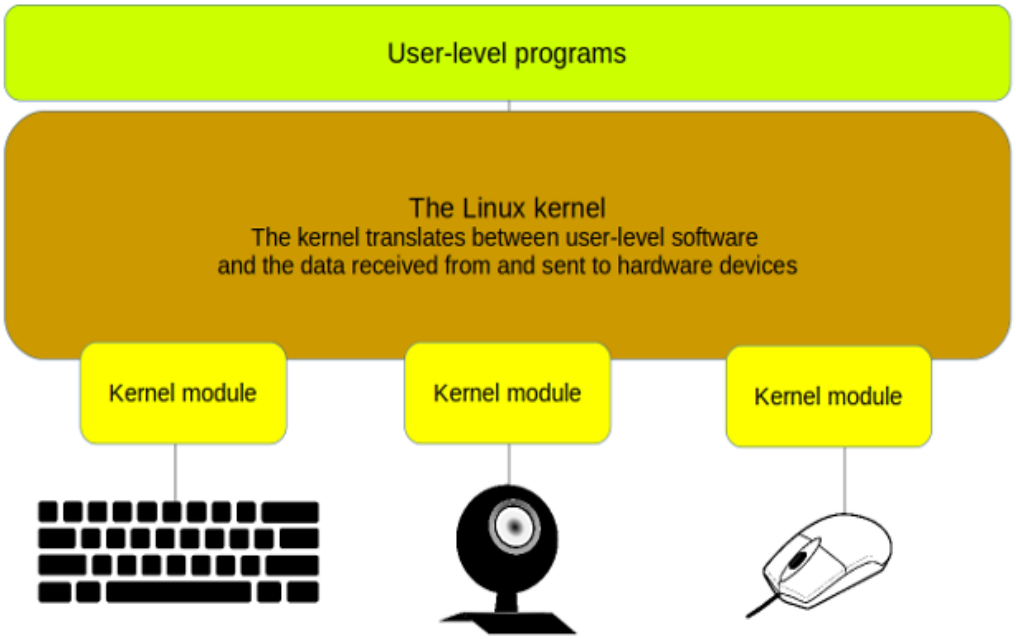
# What is Kernel Modules in Linux

- A loadable kernel module (LKM) is a mechanism for adding code to, or removing code from, the Linux kernel at run time
  - Without module, if you want to add code to a Linux kernel, the most basic way to do that is to add some source codes or files to the kernel source tree and recompile the kernel
  - With module, you can add code to the Linux kernel while it is running
    - A chunk of code that you add in this way is called a **loadable kernel module**

# What is Kernel Modules in Linux

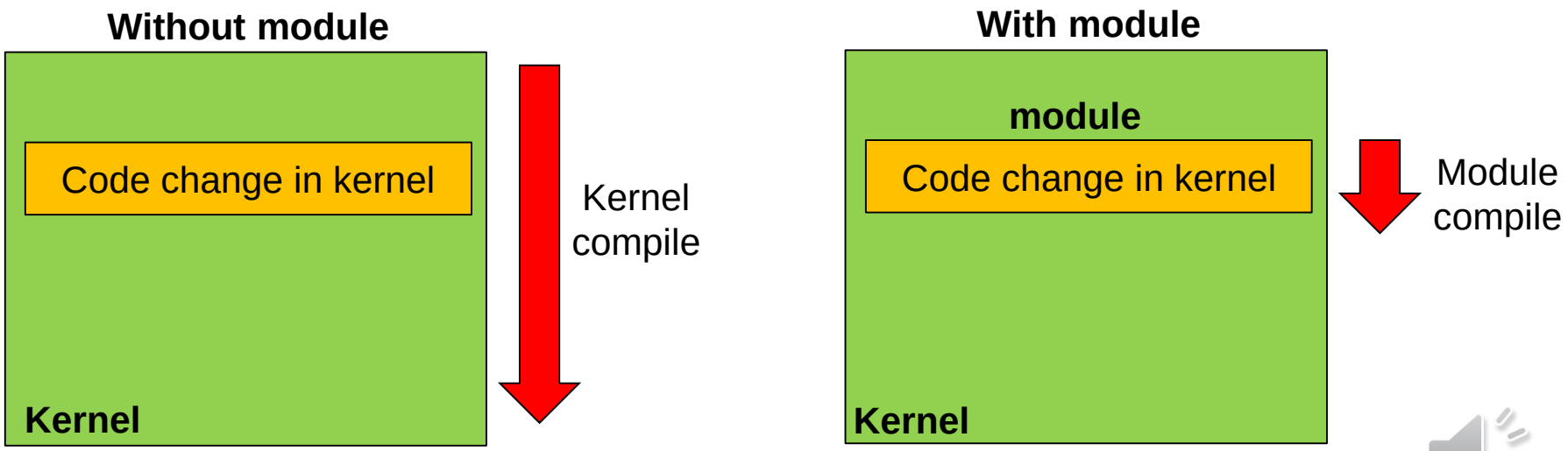
- They are ideal for device drivers, enabling the kernel to communicate with the hardware
  - A module can be represented by a file with a .ko (kernel object) extension in the **/lib/modules/** directory



# Loadable Kernel Module

- **Summary**
  - Without loadable kernel modules, developers rebuild and reboot the base kernel every time they require new functionality
    - The compile time is maximized
  - With loadable kernel modules, developers only rebuild the modules, and most OSs support loadable kernel modules
    - The compile time is minimized



# Major Module Commands

- **lsmod**
  - The lsmod command can be used to list the modules that are currently installed into the kernel
  - The result is a list of modules by name, including the memory size and usage of the module
    - **"Used by"** (usage) means how many other modules use this module
    - An example output of lsmod is shown here:

```
$ lsmod
Module                  Size  Used by
xor                    24576  1 btrfs
zstd_compress         163840  1 btrfs
raid6_pq              114688  1 btrfs
libcrc32c              16384  2 btrfs,xfs
```

# Major Module Commands

- **insmod**
  - The insmod (or insert module) command can be used insert a module into the kernel
    - It allows the user to load kernel modules at runtime to extend the kernel functionalities
    - This command insert the kernel object file (.ko) into the kernel
    - Example:
      - **insmod my_module.ko**

# Major Module Commands

- **rmmod**
  - rmmod (or remove module) command unloads loadable modules from the running kernel
  - It tries to unload a set of modules from the kernel with the restriction that they are not in use and that they are not referred to by other modules
  - If more than one module is named on the command line, the modules will be removed in the given order
    - This supports unloading of stacked modules
  - Exmaple:
    - **rmmod my_module.ko or rmmod my_module**

# Major Module Commands

- **Getting module information**
  - modinfo command in Linux system is used to display the information about a Linux Kernel module.
  - This command extracts the information from the Linux kernel modules given on the command line.
  - Example:
    - **Modinfo my_module.ko**

```
root@syslab:/home/ysson/add_module# modinfo hello_module.ko
filename:       /home/ysson/add_module/hello_module.ko
license:        GPL
srcversion:     DA8230BF9E162BF365F766C
```

# How to make and compile kernel module

- Source code of a module can be out of the kernel source tree
- Put a makefile in the module source directory
- After compilation, the compiled module is the file with .ko extension

# How to make and compile kernel module

- **Create Makefile**
  - $vi Makefile

```
#--------- Makefile ---------#

obj-m          := hello_module.o

KERNEL_DIR     := /lib/modules/$(shell uname -r)/build
PWD            := $(shell pwd)

default :
	$(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) modules
clean :
	$(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) clean
```

  - $make

# How to make and compile kernel module

- **Writing a Simple Kernel Module**

```
#--------- hello_module.c ---------#
#include <linux/kernel.h> //Needed by all modules
#include <linux/module.h> //Needed for KERN_ALERT
#include <linux/init.h> //Needed for the macros

int __init hello_module_init(void)
{
        printk("Hello Module!\n");
        return 0;
}

void __exit hello_module_cleanup(void)
{
        printk("Bye Module!\n");
}

module_init(hello_module_init);
module_exit(hello_module_cleanup);
MODULE_LICENSE("GPL");
```

# How to make and compile kernel module

- **Module initialization and exit**
  - **module_init**(hello_module_init)
    - Initialization entry point
    - Function (hello_module_init) to be run at module insertion
    - hello_module_init() will be called at loading the module
  - **module_exit**(hello_module_cleanup)
    - Exit entry point
    - Function (hello_module_cleanup) to be run at module remove
    - hello_module_cleanup() will be called at unloading the module

# How to make and compile kernel module

- **Kernel modules must have at least two functions**
  - A “start” (initialization) function (e.g., hello_module_init()) which is called when the module is insmoded into the kernel
  - An “end” (cleanup) function (e.g., hello_module_cleanup()) which is called when the module is rmmoded from the kernel

# How to make and compile kernel module

- **Launching a Kernel module**
  - Needs root privileges because you are executing kernel code
  - Loading a kernel module with **insmod**
    - **insmod hello_module.ko**
    - Module is loaded and init function is executed
  - Note that a module is compiled against a specific kernel version and will not load on another kernel

# How to make and compile kernel module

- **Launching a Kernel module**
  - Remove the module with rmmod
    - **rmmod hello_module** or **rmmod hello_module.ko**
    - Module exit function is called before unloading