Kernel Module Programming

Practitioner: Mobin Kheibary

Student Number: [994421017]

4 Hello World

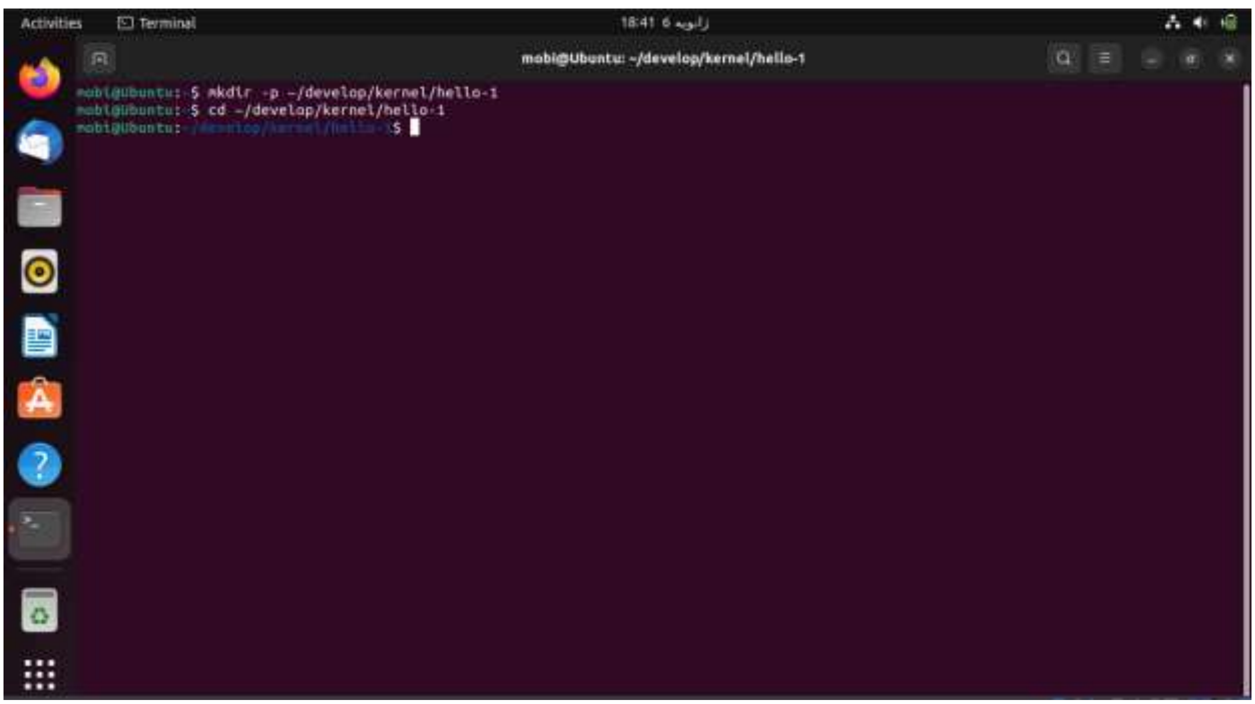
4.1 The Simplest Module

Most people learning programming start out with some sort of "*hello world*" example. I don't know what happens to people who break with this tradition, but I think it is safer not to find out. We will start with a series of hello world programs that demonstrate the different aspects of the basics of writing a kernel module.

Here is the simplest module possible.

Make a test directory:

```
mkdir -p ~/develop/kernel/hello-1
cd ~/develop/kernel/hello-1
```



Paste this into your favorite editor and save it as hello-1.c:

```c
/*
 * hello-1.c - The simplest kernel module.
 */
#include <linux/kernel.h> /* Needed for pr_info() */
#include <linux/module.h> /* Needed by all modules */

int init_module(void)
{
    pr_info("Hello world 1.\n");

    /* A non 0 return means init_module failed; module can't be loaded. */
    return 0;
}

void cleanup_module(void)
{
    pr_info("Goodbye world 1.\n");
}

MODULE_LICENSE("GPL");
```

Now you will need a Makefile. If you copy and paste this, change the indentation to use tabs, not spaces.

```
obj-m += hello-1.o

PWD := $(CURDIR)

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

In Makefile, $(CURDIR) can set to the absolute pathname of the current working directory(after all -C options are processed, if any). See more about CURDIR in GNU make manual.

And finally, just run make directly.

```
make
```

If there is no PWD := $(CURDIR) statement in Makefile, then it may not compile correctly with sudo make. Because some environment variables are specified by the security policy, they can't be inherited. The default security policy is sudoers. In the sudoers security policy, env_reset is enabled by default, which restricts environment variables. Specifically, path variables are not retained from the user environment, they are set to default values (For more information see: sudoers manual). You can see the environment variable settings by:

$ sudo -s

sudo -V

Here is a simple Makefile as an example to demonstrate the problem mentioned above.

```
all:
        echo $(PWD)
```

Then, we can use -p flag to print out the environment variable values from the Makefile.

$ make -p | grep PWD

PWD = /home/ubuntu/temp

OLDPWD = /home/ubuntu

echo $(PWD)

The PWD variable won't be inherited with sudo.

$ sudo make -p | grep PWD

echo $(PWD)

However, there are three ways to solve this problem.

1. You can use the -E flag to temporarily preserve them.

```
$ sudo -E make -p | grep PWD
PWD = /home/ubuntu/temp
OLDPWD = /home/ubuntu
echo $(PWD)
```

2. You can set the env_reset disabled by editing the /etc/sudoers with root and visudo.

```
## sudoers file.
##
...
Defaults env_reset
## Change env_reset to !env_reset in previous line to keep all environment variables
```

Then execute env and sudo env individually.

```
# disable the env_reset
echo "user:" > non-env_reset.log; env >> non-env_reset.log
echo "root:" >> non-env_reset.log; sudo env >> non-env_reset.log
# enable the env_reset
echo "user:" > env_reset.log; env >> env_reset.log
echo "root:" >> env_reset.log; sudo env >> env_reset.log
```

You can view and compare these logs to find differences between env_reset and !env_reset.

3. You can preserve environment variables by appending them to env_keep in /etc/sudoers.

```
Defaults env_keep += "PWD"
```

After applying the above change, you can check the environment variable settings by:

`$ sudo -s`

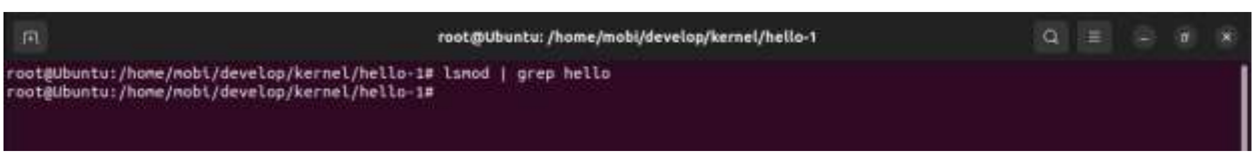
sudo -V

If all goes smoothly you should then find that you have a compiled hello-1.ko module. You can find info on it with the command:

```
modinfo hello-1.ko
```

At this point the command:

```
sudo lsmod | grep hello
```

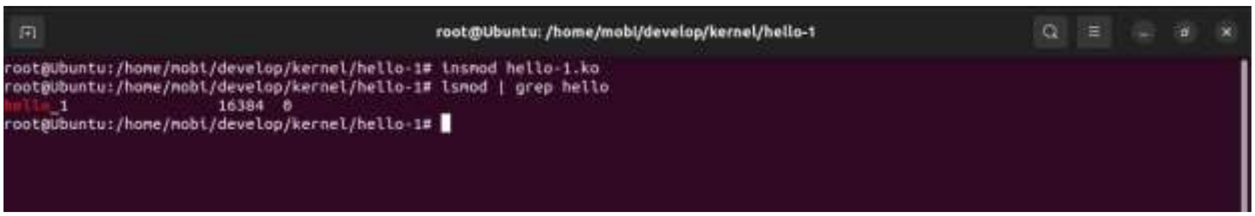


should return nothing. You can try loading your shiny new module with:

```
sudo insmod hello-1.ko
```

The dash character will get converted to an underscore, so when you again try:

```
sudo lsmod | grep hello
```

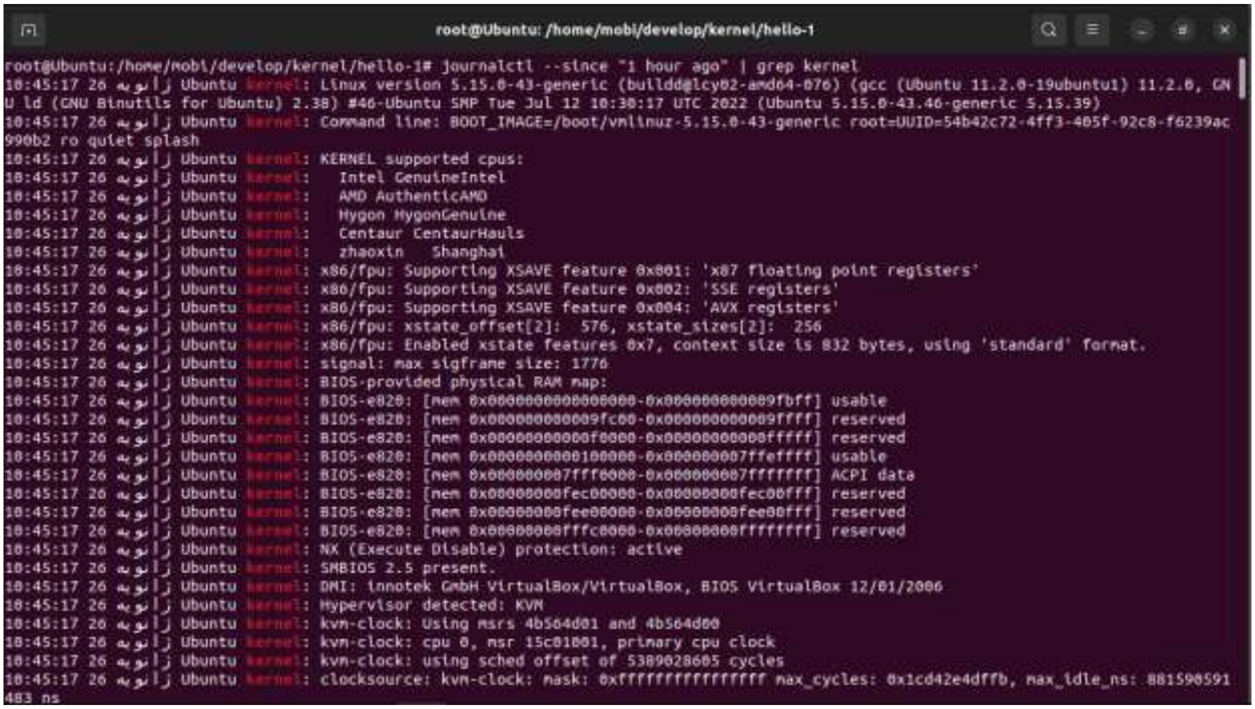


you should now see your loaded module. It can be removed again with:

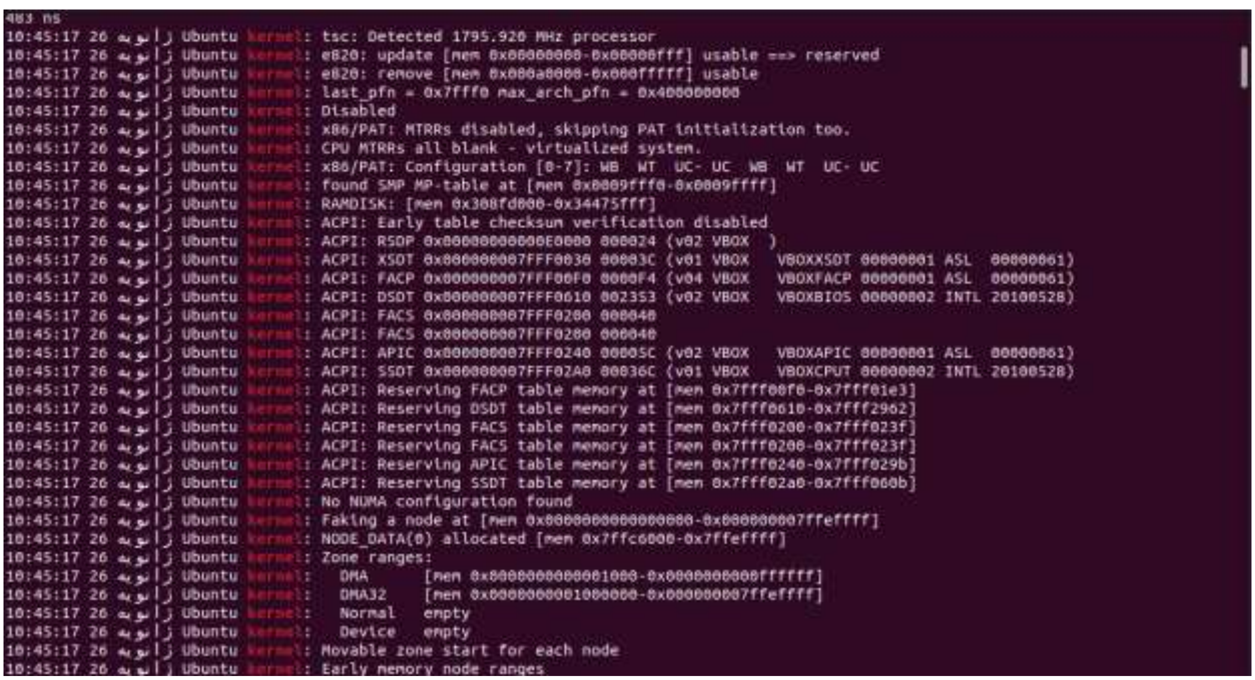
```
sudo rmmod hello_1
```

Notice that the dash was replaced by an underscore. To see what just happened in the logs:

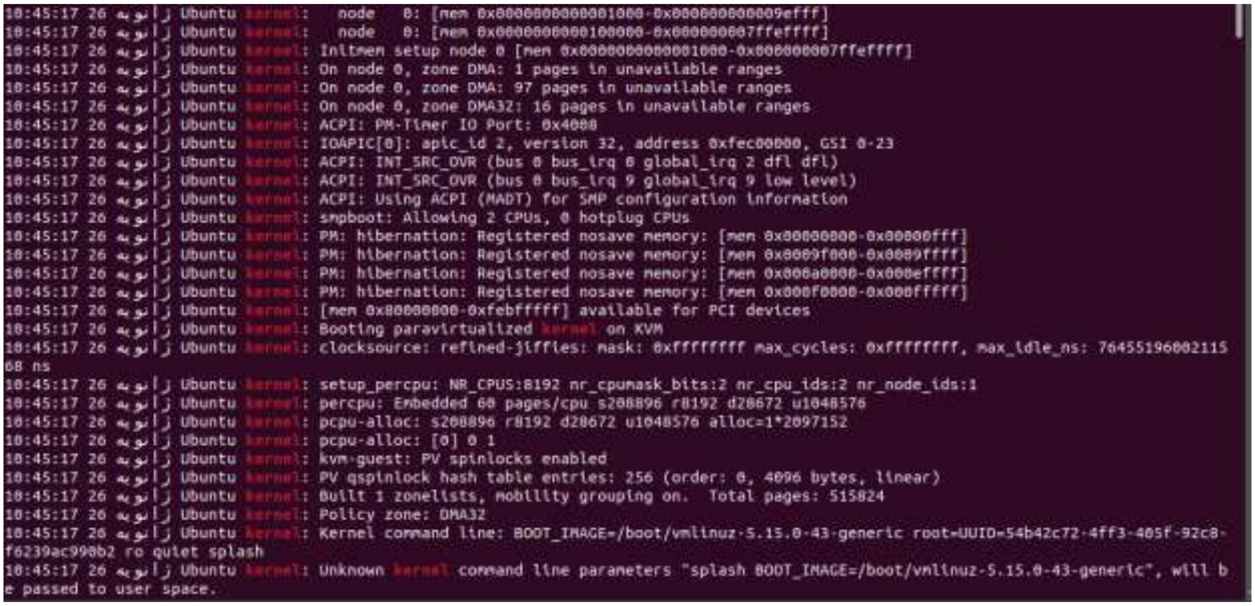
```
sudo journalctl --since "1 hour ago" | grep kernel
```



```
root@Ubuntu: /home/mobi/develop/kernel/hello-1
root@Ubuntu:/home/mobi/develop/kernel/hello-1# journalctl --since "1 hour ago" | grep kernel
10:45:17 26 ژانویه Ubuntu kernel: Linux version 5.15.0-43-generic (buildd@lcy02-amd64-076) (gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #46-Ubuntu SMP Tue Jul 12 10:30:17 UTC 2022 (Ubuntu 5.15.0-43.46-generic 5.15.39)
10:45:17 26 ژانویه Ubuntu kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-43-generic root=UUID=54b42c72-4ff3-405f-92c8-f6239ac990b2 ro quiet splash
10:45:17 26 ژانویه Ubuntu kernel: KERNEL supported cpus:
10:45:17 26 ژانویه Ubuntu kernel:   Intel GenuineIntel
10:45:17 26 ژانویه Ubuntu kernel:   AMD AuthenticAMD
10:45:17 26 ژانویه Ubuntu kernel:   Hygon HygonGenuine
10:45:17 26 ژانویه Ubuntu kernel:   Centaur CentaurHauls
10:45:17 26 ژانویه Ubuntu kernel:   zhaoxin   Shanghai
10:45:17 26 ژانویه Ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
10:45:17 26 ژانویه Ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
10:45:17 26 ژانویه Ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
10:45:17 26 ژانویه Ubuntu kernel: x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
10:45:17 26 ژانویه Ubuntu kernel: x86/fpu: Enabled xstate features 0x7, context size is 832 bytes, using 'standard' format.
10:45:17 26 ژانویه Ubuntu kernel: signal: max sigframe size: 1776
10:45:17 26 ژانویه Ubuntu kernel: BIOS-provided physical RAM map:
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x0000000000100000-0x000000007ffeffff] usable
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x000000007fff0000-0x000000007fffffff] ACPI data
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x00000000fec00000-0x00000000fec00fff] reserved
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x00000000fee00000-0x00000000fee00fff] reserved
10:45:17 26 ژانویه Ubuntu kernel: BIOS-e820: [mem 0x00000000fffc0000-0x00000000ffffffff] reserved
10:45:17 26 ژانویه Ubuntu kernel: NX (Execute Disable) protection: active
10:45:17 26 ژانویه Ubuntu kernel: SMBIOS 2.5 present.
10:45:17 26 ژانویه Ubuntu kernel: DMI: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
10:45:17 26 ژانویه Ubuntu kernel: Hypervisor detected: KVM
10:45:17 26 ژانویه Ubuntu kernel: kvm-clock: Using msrs 4b564d01 and 4b564d00
10:45:17 26 ژانویه Ubuntu kernel: kvm-clock: cpu 0, msr 15c01001, primary cpu clock
10:45:17 26 ژانویه Ubuntu kernel: kvm-clock: using sched offset of 5389028605 cycles
10:45:17 26 ژانویه Ubuntu kernel: clocksource: kvm-clock: mask: 0xffffffffffffffff max_cycles: 0x1cd42e4dffb, max_idle_ns: 881590591483 ns
```



```
483 ns
10:45:17 26 ژانویه Ubuntu kernel: tsc: Detected 1795.920 MHz processor
10:45:17 26 ژانویه Ubuntu kernel: e820: update [mem 0x00000000-0x00000fff] usable ==> reserved
10:45:17 26 ژانویه Ubuntu kernel: e820: remove [mem 0x000a0000-0x000fffff] usable
10:45:17 26 ژانویه Ubuntu kernel: last_pfn = 0x7fff0 max_arch_pfn = 0x400000000
10:45:17 26 ژانویه Ubuntu kernel: Disabled
10:45:17 26 ژانویه Ubuntu kernel: x86/PAT: MTRRs disabled, skipping PAT initialization too.
10:45:17 26 ژانویه Ubuntu kernel: CPU MTRRs all blank - virtualized system.
10:45:17 26 ژانویه Ubuntu kernel: x86/PAT: Configuration [0-7]: WB  WT  UC- UC  WB  WT  UC- UC
10:45:17 26 ژانویه Ubuntu kernel: found SMP MP-table at [mem 0x0009fff0-0x0009ffff]
10:45:17 26 ژانویه Ubuntu kernel: RAMDISK: [mem 0x308fd000-0x34475fff]
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Early table checksum verification disabled
10:45:17 26 ژانویه Ubuntu kernel: ACPI: RSDP 0x00000000000E0000 000024 (v02 VBOX  )
10:45:17 26 ژانویه Ubuntu kernel: ACPI: XSDT 0x000000007FFF0030 00003C (v01 VBOX   VBOXXSDT 00000001 ASL  00000061)
10:45:17 26 ژانویه Ubuntu kernel: ACPI: FACP 0x000000007FFF00F0 0000F4 (v04 VBOX   VBOXFACP 00000001 ASL  00000061)
10:45:17 26 ژانویه Ubuntu kernel: ACPI: DSDT 0x000000007FFF0610 002353 (v02 VBOX   VBOXBIOS 00000002 INTL 20100528)
10:45:17 26 ژانویه Ubuntu kernel: ACPI: FACS 0x000000007FFF0200 000040
10:45:17 26 ژانویه Ubuntu kernel: ACPI: FACS 0x000000007FFF0200 000040
10:45:17 26 ژانویه Ubuntu kernel: ACPI: APIC 0x000000007FFF0240 00005C (v02 VBOX   VBOXAPIC 00000001 ASL  00000061)
10:45:17 26 ژانویه Ubuntu kernel: ACPI: SSDT 0x000000007FFF02A0 00036C (v01 VBOX   VBOXCPUT 00000002 INTL 20100528)
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Reserving FACP table memory at [mem 0x7fff00f0-0x7fff01e3]
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Reserving DSDT table memory at [mem 0x7fff0610-0x7fff2962]
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Reserving FACS table memory at [mem 0x7fff0200-0x7fff023f]
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Reserving FACS table memory at [mem 0x7fff0200-0x7fff023f]
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Reserving APIC table memory at [mem 0x7fff0240-0x7fff029b]
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Reserving SSDT table memory at [mem 0x7fff02a0-0x7fff060b]
10:45:17 26 ژانویه Ubuntu kernel: No NUMA configuration found
10:45:17 26 ژانویه Ubuntu kernel: Faking a node at [mem 0x0000000000000000-0x000000007ffeffff]
10:45:17 26 ژانویه Ubuntu kernel: NODE_DATA(0) allocated [mem 0x7ffc6000-0x7ffeffff]
10:45:17 26 ژانویه Ubuntu kernel: Zone ranges:
10:45:17 26 ژانویه Ubuntu kernel:   DMA      [mem 0x0000000000001000-0x0000000000ffffff]
10:45:17 26 ژانویه Ubuntu kernel:   DMA32    [mem 0x0000000001000000-0x000000007ffeffff]
10:45:17 26 ژانویه Ubuntu kernel:   Normal   empty
10:45:17 26 ژانویه Ubuntu kernel:   Device   empty
10:45:17 26 ژانویه Ubuntu kernel: Movable zone start for each node
10:45:17 26 ژانویه Ubuntu kernel: Early memory node ranges
```



```
10:45:17 26 ژانویه Ubuntu kernel:   node   0: [mem 0x0000000000001000-0x000000000009efff]
10:45:17 26 ژانویه Ubuntu kernel:   node   0: [mem 0x0000000000100000-0x000000007ffeffff]
10:45:17 26 ژانویه Ubuntu kernel: Initmem setup node 0 [mem 0x0000000000001000-0x000000007ffeffff]
10:45:17 26 ژانویه Ubuntu kernel: On node 0, zone DMA: 1 pages in unavailable ranges
10:45:17 26 ژانویه Ubuntu kernel: On node 0, zone DMA: 97 pages in unavailable ranges
10:45:17 26 ژانویه Ubuntu kernel: On node 0, zone DMA32: 16 pages in unavailable ranges
10:45:17 26 ژانویه Ubuntu kernel: ACPI: PM-Timer IO Port: 0x4008
10:45:17 26 ژانویه Ubuntu kernel: IOAPIC[0]: apic_id 2, version 32, address 0xfec00000, GSI 0-23
10:45:17 26 ژانویه Ubuntu kernel: ACPI: INT_SRC_OVR (bus 0 bus_irq 0 global_irq 2 dfl dfl)
10:45:17 26 ژانویه Ubuntu kernel: ACPI: INT_SRC_OVR (bus 0 bus_irq 9 global_irq 9 low level)
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Using ACPI (MADT) for SMP configuration information
10:45:17 26 ژانویه Ubuntu kernel: smpboot: Allowing 2 CPUs, 0 hotplug CPUs
10:45:17 26 ژانویه Ubuntu kernel: PM: hibernation: Registered nosave memory: [mem 0x00000000-0x00000fff]
10:45:17 26 ژانویه Ubuntu kernel: PM: hibernation: Registered nosave memory: [mem 0x0009f000-0x0009ffff]
10:45:17 26 ژانویه Ubuntu kernel: PM: hibernation: Registered nosave memory: [mem 0x000a0000-0x000effff]
10:45:17 26 ژانویه Ubuntu kernel: PM: hibernation: Registered nosave memory: [mem 0x000f0000-0x000fffff]
10:45:17 26 ژانویه Ubuntu kernel: [mem 0x80000000-0xfebfffff] available for PCI devices
10:45:17 26 ژانویه Ubuntu kernel: Booting paravirtualized kernel on KVM
10:45:17 26 ژانویه Ubuntu kernel: clocksource: refined-jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 76455196002115
68 ns
10:45:17 26 ژانویه Ubuntu kernel: setup_percpu: NR_CPUS:8192 nr_cpumask_bits:2 nr_cpu_ids:2 nr_node_ids:1
10:45:17 26 ژانویه Ubuntu kernel: percpu: Embedded 60 pages/cpu s208896 r8192 d28672 u1048576
10:45:17 26 ژانویه Ubuntu kernel: pcpu-alloc: s208896 r8192 d28672 u1048576 alloc=1*2097152
10:45:17 26 ژانویه Ubuntu kernel: pcpu-alloc: [0] 0 1
10:45:17 26 ژانویه Ubuntu kernel: kvm-guest: PV spinlocks enabled
10:45:17 26 ژانویه Ubuntu kernel: PV qspinlock hash table entries: 256 (order: 0, 4096 bytes, linear)
10:45:17 26 ژانویه Ubuntu kernel: Built 1 zonelists, mobility grouping on.  Total pages: 515824
10:45:17 26 ژانویه Ubuntu kernel: Policy zone: DMA32
10:45:17 26 ژانویه Ubuntu kernel: Kernel command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-43-generic root=UUID=54b42c72-4ff3-405f-92c8-
f6239ac990b2 ro quiet splash
10:45:17 26 ژانویه Ubuntu kernel: Unknown kernel command line parameters "splash BOOT_IMAGE=/boot/vmlinuz-5.15.0-43-generic", will b
e passed to user space.
```

```
10:45:17 26 ژانویه Ubuntu kernel: Dentry cache hash table entries: 262144 (order: 9, 2097152 bytes, linear)
10:45:17 26 ژانویه Ubuntu kernel: Inode-cache hash table entries: 131072 (order: 8, 1048576 bytes, linear)
10:45:17 26 ژانویه Ubuntu kernel: mem auto-init: stack:off, heap alloc:on, heap free:off
10:45:17 26 ژانویه Ubuntu kernel: Memory: 1955284K/2096696K available (16393K kernel code, 4382K rwdata, 10800K rodata, 2904K init,
4844K bss, 141152K reserved, 0K cma-reserved)
10:45:17 26 ژانویه Ubuntu kernel: random: get_random_u64 called from kmem_cache_open+0x2b/0x320 with crng_init=0
10:45:17 26 ژانویه Ubuntu kernel: SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=2, Nodes=1
10:45:17 26 ژانویه Ubuntu kernel: Kernel/User page tables isolation: enabled
10:45:17 26 ژانویه Ubuntu kernel: ftrace: allocating 50457 entries in 198 pages
10:45:17 26 ژانویه Ubuntu kernel: ftrace: allocated 198 pages with 4 groups
10:45:17 26 ژانویه Ubuntu kernel: rcu: Hierarchical RCU implementation.
10:45:17 26 ژانویه Ubuntu kernel: rcu:         RCU restricting CPUs from NR_CPUS=8192 to nr_cpu_ids=2.
10:45:17 26 ژانویه Ubuntu kernel:         Rude variant of Tasks RCU enabled.
10:45:17 26 ژانویه Ubuntu kernel:         Tracing variant of Tasks RCU enabled.
10:45:17 26 ژانویه Ubuntu kernel: rcu: RCU calculated value of scheduler-enlistment delay is 25 jiffies.
10:45:17 26 ژانویه Ubuntu kernel: rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=2
10:45:17 26 ژانویه Ubuntu kernel: NR_IRQS: 524544, nr_irqs: 440, preallocated irqs: 16
10:45:17 26 ژانویه Ubuntu kernel: random: crng done (trusting CPU's manufacturer)
10:45:17 26 ژانویه Ubuntu kernel: Console: colour VGA+ 80x25
10:45:17 26 ژانویه Ubuntu kernel: printk: console [tty0] enabled
10:45:17 26 ژانویه Ubuntu kernel: ACPI: Core revision 20210730
10:45:17 26 ژانویه Ubuntu kernel: APIC: Switch to symmetric I/O mode setup
10:45:17 26 ژانویه Ubuntu kernel: x2apic enabled
10:45:17 26 ژانویه Ubuntu kernel: Switched APIC routing to physical x2apic.
10:45:17 26 ژانویه Ubuntu kernel: ..TIMER: vector=0x30 apic1=0 pin1=2 apic2=-1 pin2=-1
10:45:17 26 ژانویه Ubuntu kernel: clocksource: tsc-early: mask: 0xffffffffffffffff max_cycles: 0x19e31b7216c, max_idle_ns: 440795249
889 ns
10:45:17 26 ژانویه Ubuntu kernel: Calibrating delay loop (skipped) preset value.. 3591.84 BogoMIPS (lpj=7183680)
10:45:17 26 ژانویه Ubuntu kernel: pid_max: default: 32768 minimum: 301
10:45:17 26 ژانویه Ubuntu kernel: LSM: Security Framework initializing
10:45:17 26 ژانویه Ubuntu kernel: landlock: Up and running.
```

And So on…

You now know the basics of creating, compiling, installing and removing modules. Now for more of a description of how this module works.
Kernel modules must have at least two functions: a "start" (initialization) function called init_module() which is called when the module is insmoded into the kernel, and an "end" (cleanup) function called cleanup_module() which is called just before it is removed from the kernel. Actually, things have changed starting with kernel 2.3.13. You can now use whatever name you like for the start and end functions of a module, and you will learn how to do this in Section 4.2. In fact, the new method is the preferred method. However, many people still use init_module() and cleanup_module() for their start and end functions.
Typically, init_module() either registers a handler for something with the kernel, or it replaces one of the kernel functions with its own code (usually code to do something and then call the original function). The cleanup_module() function is supposed to undo whatever init_module() did, so the module can be unloaded safely.
Lastly, every kernel module needs to include <linux/module.h>. We needed to include <linux/kernel.h> only for the macro expansion for the pr_alert() log level, which you'll learn about in Section 2.
1. A point about coding style. Another thing which may not be immediately obvious to anyone getting started with kernel programming is that indentation within your code should be using tabs and not spaces. It is one of the coding conventions of the kernel. You may not like it, but you'll need to get used to it if you ever submit a patch upstream.2. Introducing print macros. In the beginning there was printk, usually followed by a priority such as KERN_INFO or KERN_DEBUG. More recently this

can also be expressed in abbreviated form using a set of print macros,

such as pr_info and pr_debug. This just saves some mindless keyboard bashing and looks a bit neater. They can be found within include/linux/printk.h. Take time to read through the available priority

macros.

3. About Compiling. Kernel modules need to be compiled a bit differently

from regular userspace apps. Former kernel versions required us to care

much about these settings, which are usually stored in Makefiles. Although hierarchically organized, many redundant settings accumulated in

sublevel Makefiles and made them large and rather difficult to maintain.

Fortunately, there is a new way of doing these things, called kbuild, and

the build process for external loadable modules is now fully integrated into

the standard kernel build mechanism. To learn more on how to compile

modules which are not part of the official kernel (such as all the examples

you will find in this guide), see file Documentation/kbuild/modules.rst.

Additional details about Makefiles for kernel modules are available in Documentation/kbuild/makefiles.rst. Be sure to read this and the related files

before starting to hack Makefiles. It will probably save you lots of work.

Here is another exercise for the reader. See that comment above

the return statement in init_module()? Change the return

value to something negative, recompile and load the module

again. What happens?

4.2 Hello and Goodbye

In early kernel versions you had to use the init_module and cleanup_module

functions, as in the first hello world example, but these days you can name those

anything you want by using the module_init and module_exit macros. These macros are defined in include/linux/module.h. The only requirement is that your init and cleanup functions must be defined before calling those macros, otherwise, you'll get compilation errors. Here is an example of this technique:

```c
/*
 * hello-2.c - Demonstrating the module_init() and module_exit() macros.
 * This is preferred over using init_module() and cleanup_module().
 */
#include <linux/init.h> /* Needed for the macros */
#include <linux/kernel.h> /* Needed for pr_info() */
#include <linux/module.h> /* Needed by all modules */

static int __init hello_2_init(void)
{
    pr_info("Hello, world 2\n");
    return 0;
}

static void __exit hello_2_exit(void)
{
    pr_info("Goodbye, world 2\n");
}

module_init(hello_2_init);
module_exit(hello_2_exit);

MODULE_LICENSE("GPL");
```

So now we have two real kernel modules under our belt. Adding another module is as simple as this:

```makefile
obj-m += hello-1.o
obj-m += hello-2.o

PWD := $(CURDIR)

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```c
/*
 * hello-2.c - Demonstrating the module_init() and module_exit() macros.
 * This is preferred over using init_module() and cleanup_module().
 */
#include <linux/init.h> /* Needed for the macros */
#include <linux/kernel.h> /* Needed for pr_info() */
#include <linux/module.h> /* Needed by all modules */

static int __init hello_2_init(void)
{
        pr_info("Hello, world 2.\n");
        return 0;
}

static void __exit hello_2_exit(void)
{
        pr_info("Goodbye, world2.\n");
}

module_init(hello_2_init);
module_exit(hello_2_exit);

MODULE_LICENSE("GPL");
```

```makefile
obj-m += hello-1.o
obj-m += hello-2.o

PWD := $(CURDIR)

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Now have a look at drivers/char/Makefile for a real-world example. As you can see, some things got hardwired into the kernel (obj-y) but where have all those obj-m gone? Those familiar with shell scripts will easily be able to spot them. For those who are not, the obj-$(CONFIG_FOO) entries you see everywhere expand into obj-y or obj-m, depending on whether the CONFIG_FOO variable has been set to y or m. While we are at it, those were exactly the kind of variables that you have set in the .config file in the top-level directory of Linux kernel source tree, the last time when you said make menuconfig or something like that.

4.3 The __init and __exit Macros

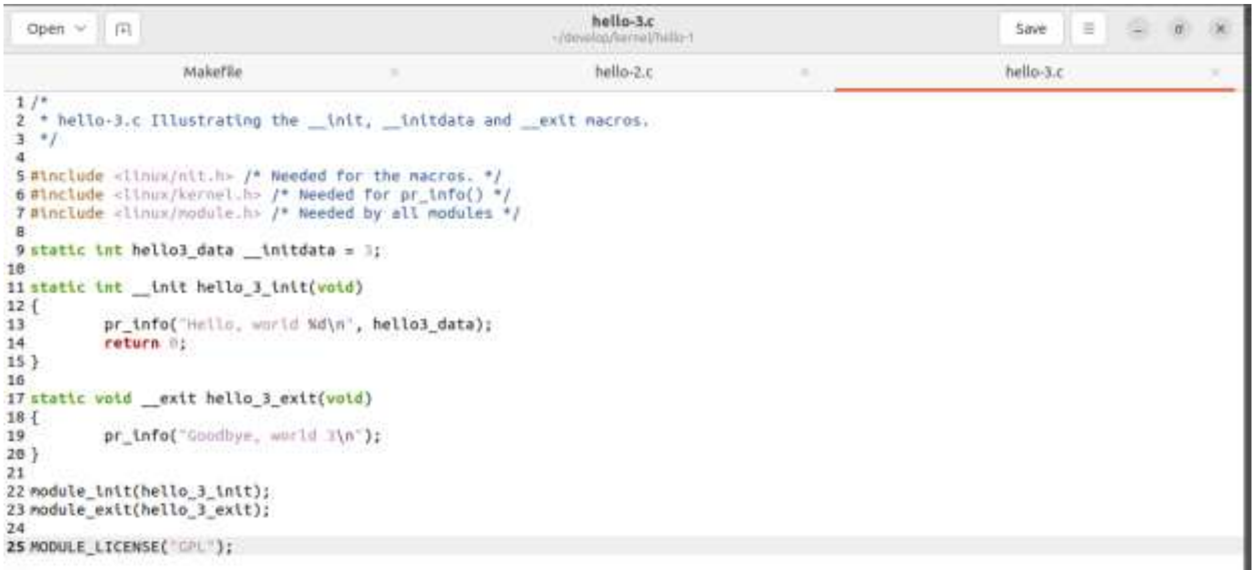
The __init macro causes the init function to be discarded and its memory freed once the init function finishes for built-in drivers, but not loadable modules. If you think about when the init function is invoked, this makes perfect sense.

There is also an __initdata which works similarly to __init but for init variables rather than functions.

The __exit macro causes the omission of the function when the module is built into the kernel, and like __init, has no effect for loadable modules. Again, if you consider when the cleanup function runs, this makes complete sense; built-in drivers do not need a cleanup function, while loadable modules do.

These macros are defined in include/linux/init.h and serve to free up kernel memory. When you boot your kernel and see something like Freeing unused kernel memory: 236k freed, this is precisely what the kernel is freeing.

```c
/*
 * hello-3.c - Illustrating the __init, __initdata and __exit macros.
 */
#include <linux/init.h> /* Needed for the macros */
#include <linux/kernel.h> /* Needed for pr_info() */
#include <linux/module.h> /* Needed by all modules */

static int hello3_data __initdata = 3;

static int __init hello_3_init(void)
{
    pr_info("Hello, world %d\n", hello3_data);
    return 0;
}

static void __exit hello_3_exit(void)
{
    pr_info("Goodbye, world 3\n");
}

module_init(hello_3_init);
module_exit(hello_3_exit);

MODULE_LICENSE("GPL");
```



```
/*
 * hello-3.c Illustrating the __init, __initdata and __exit macros.
 */

#include <linux/nit.h> /* Needed for the macros. */
#include <linux/kernel.h> /* Needed for pr_info() */
#include <linux/module.h> /* Needed by all modules */

static int hello3_data __initdata = 3;

static int __init hello_3_init(void)
{
        pr_info("Hello, world %d\n", hello3_data);
        return 0;
}

static void __exit hello_3_exit(void)
{
        pr_info("Goodbye, world 3\n");
}

module_init(hello_3_init);
module_exit(hello_3_exit);

MODULE_LICENSE("GPL");
```

4.4 Licensing and Module Documentation

Honestly, who loads or even cares about proprietary modules? If you do then you might have seen something like this:

```
$ sudo insmod xxxxxx.ko
```

loading out-of-tree module taints kernel.

module license 'unspecified' taints kernel.

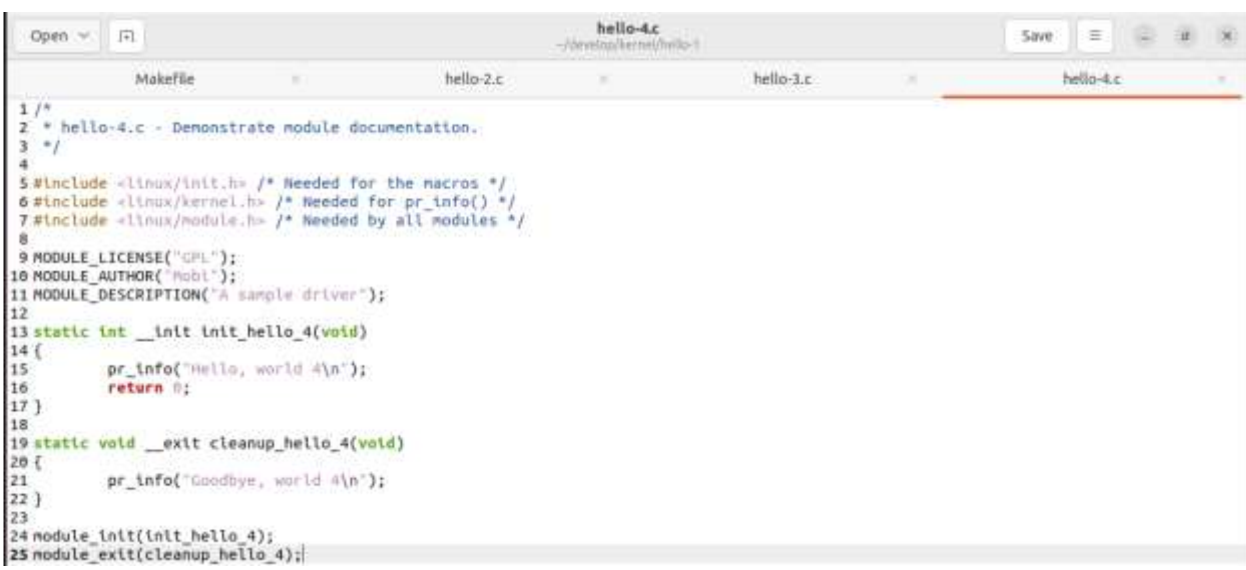
You can use a few macros to indicate the license for your module. Some examples are "GPL", "GPL v2", "GPL and additional rights", "Dual BSD/GPL", "Dual MIT/GPL", "Dual MPL/GPL" and "Proprietary". They are defined within include/linux/module.h.

To reference what license you're using a macro is available called MODULE_LICENSE.

This and a few other macros describing the module are illustrated in the below example.

```c
/*
 * hello-4.c - Demonstrates module documentation.
 */
#include <linux/init.h> /* Needed for the macros */
#include <linux/kernel.h> /* Needed for pr_info() */
#include <linux/module.h> /* Needed by all modules */
```

```c

MODULE_LICENSE("GPL");
MODULE_AUTHOR("LKMPG");
MODULE_DESCRIPTION("A sample driver");

static int __init init_hello_4(void)
{
    pr_info("Hello, world 4\n");
    return 0;
}

static void __exit cleanup_hello_4(void)
{
    pr_info("Goodbye, world 4\n");
}

module_init(init_hello_4);
module_exit(cleanup_hello_4);
```



4.5 Passing Command Line Arguments to a Module

Modules can take command line arguments, but not with the argc/argv you might be used to.

To allow arguments to be passed to your module, declare the variables that will take the values of the command line arguments as global and then use the module_param() macro, (defined in include/linux/moduleparam.h) to set the mechanism up. At runtime, insmod will fill the variables with any command line arguments that are given, like insmod mymodule.ko myvariable=5. The variable declarations and macros should be placed at the beginning of the module for clarity. The example code should clear up my admittedly lousy explanation.

The module_param() macro takes 3 arguments: the name of the variable, its type and permissions for the corresponding file in sysfs. Integer types can be signed as usual or unsigned. If you'd like to use arrays of integers or strings see module_param_array() and module_param_string().

```
int myint = 3;
module_param(myint, int, 0);
```

Arrays are supported too, but things are a bit different now than they were in the olden days. To keep track of the number of parameters you need to pass a pointer to a count variable as third parameter. At your option, you could also ignore the count and pass NULL instead. We show both possibilities here:

```
int myintarray[2];
module_param_array(myintarray, int, NULL, 0); /* not interested in count */

short myshortarray[4];
int count;
module_param_array(myshortarray, short, &count, 0); /* put count into "count"
↪  variable */
```

A good use for this is to have the module variable's default values set, like a port or IO address. If the variables contain the default values, then perform autodetection (explained elsewhere). Otherwise, keep the current value. This will be made clear later on.

Lastly, there is a macro function, MODULE_PARM_DESC(), that is used to document arguments that the module can take. It takes two parameters: a variable name and a free form string describing that variable.

```c
/*
 * hello-5.c - Demonstrates command line argument passing to a module.
 */
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/stat.h>

MODULE_LICENSE("GPL");

static short int myshort = 1;
static int myint = 420;
static long int mylong = 9999;
static char *mystring = "blah";
static int myintarray[2] = { 420, 420 };
static int arr_argc = 0;

/* module_param(foo, int, 0000)
 * The first param is the parameters name.
 * The second param is its data type.
 * The final argument is the permissions bits,
 * for exposing parameters in sysfs (if non-zero) at a later stage.
 */
module_param(myshort, short, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);
MODULE_PARM_DESC(myshort, "A short integer");
module_param(myint, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
MODULE_PARM_DESC(myint, "An integer");
module_param(mylong, long, S_IRUSR);
MODULE_PARM_DESC(mylong, "A long integer");
module_param(mystring, charp, 0000);
```

```
MODULE_PARM_DESC(mylong, "A long integer");
module_param(mystring, charp, 0000);
MODULE_PARM_DESC(mystring, "A character string");

/* module_param_array(name, type, num, perm);
 * The first param is the parameter's (in this case the array's) name.
 * The second param is the data type of the elements of the array.
 * The third argument is a pointer to the variable that will store the number
 * of elements of the array initialized by the user at module loading time.
 * The fourth argument is the permission bits.
 */
module_param_array(myintarray, int, &arr_argc, 0000);
MODULE_PARM_DESC(myintarray, "An array of integers");

static int __init hello_5_init(void)
{
    int i;

    pr_info("Hello, world 5\n=============\n");
```

```
    pr_info("myint is an integer: %d\n", myint);
    pr_info("mylong is a long integer: %ld\n", mylong);
    pr_info("mystring is a string: %s\n", mystring);

    for (i = 0; i < ARRAY_SIZE(myintarray); i++)
        pr_info("myintarray[%d] = %d\n", i, myintarray[i]);

    pr_info("got %d arguments for myintarray.\n", arr_argc);
    return 0;
}

static void __exit hello_5_exit(void)
{
    pr_info("Goodbye, world 5\n");
}

module_init(hello_5_init);
module_exit(hello_5_exit);
```

I would recommend playing around with this code:

```
$ sudo insmod hello-5.ko mystring="bebop" myintarray=-1
$ sudo dmesg -t | tail -7
myshort is a short integer: 1
myint is an integer: 420
mylong is a long integer: 9999
mystring is a string: bebop
myintarray[0] = -1
myintarray[1] = 420
got 1 arguments for myintarray.
$ sudo rmmod hello-5
$ sudo dmesg -t | tail -1
Goodbye, world 5
$ sudo insmod hello-5.ko mystring="supercalifragilisticexpialidocious" myintarray=-1,-1
$ sudo dmesg -t | tail -7
myshort is a short integer: 1
myint is an integer: 420
mylong is a long integer: 9999
mystring is a string: supercalifragilisticexpialidocious
myintarray[0] = -1
myintarray[1] = -1
got 2 arguments for myintarray.
$ sudo rmmod hello-5
$ sudo dmesg -t | tail -1
```

Goodbye, world 5

$ sudo insmod hello-5.ko mylong=hello

insmod: ERROR: could not insert module hello-5.ko: Invalid parameters

Open | hello-5.c ~/develop/kernel/hello-1 | Save

Makefile | hello-2.c | hello-3.c | hello-4.c | hello-5.c

```c
/*
 * hello-5.c - Demonstrates command line arguent passing to a module.
 */

#include <linux/init.h>
#include <linux.kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/stat.h>

MODULE_LICENSE("GPL");

static short int myshort = 1;
static int myint = 420;
static long int mylong = 9999;
static char *mystring = "blah";
static int myintarray[2] = {420, 420};
static int arr_argc = 0;

/* module_param(foo, int, 0000)
 * The first param is the parameters name.
 * The second param is its data type.
 * The final arguments is the permissions bits.
 * for exposing parameters in sysfs (if non-zero) at a later stage.
 */

module_param(myshort, short, S_IRUSR | S_TWUSR | S_IRGRP | S_IWGRP);
MODULE_PARM_DESC(myshort, "A short integer");
module_param(myint, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
MODULE_PARM_DESC(myint, "An integer");
module_param(mylong, long, S_IRUSR);
MODULE_PARM_DESC(mylong, "A long integer")
```

```c
MODULE_PARM_DESC(mylong, "A long integer")
module_param(mystring, charp, 0000);
MODULE_PARM_DESC(mystring, "A character string");

/* module_param_array(name, type, num, perm);
 * The first param is the parameter's (in this case the array's) name.
 * The second param is the data type of the elements of the array.
 * The third argument is a pointer to the variable that will store the number
 * of elements of the array nitialized by the user at module loading time.
 * The fourth argument is the permission bits.
*/
module_param_array(myintarray, int, &arr_argc, 0000);
MODULE_PARM_DESC(myintarray, "An array of integers");

static int __init hello_5_init(void)
{
        int i;

        pr_info("Hello, world 5\n=============");
        pr_info("myshort is a short integer: %hd\n", myshort);
        pr_info("myint is an integer: %d\n", myint);
        pr_info("mylong is a long integer: %ld\n", mylong);
        pr_info("mystring is a string: %s\n", mystring);

        for (i = 0; i < ARRAY_SIZE(myintarray); i++)
                pr_info("myintarray[%d] = %d\n", i, myintarray[i]);

        pr_info("got %d arguments for myintarray.\n", arr_argcs);
        return 0;
}
```

C | Tab Width: 8 | Ln 69, Col 27 | INS

```c
static int __init hello_5_init(void)
{
	int i;

	pr_info("Hello, world 5\n=============");
	pr_info("myshort is a short integer: %hd\n", myshort);
	pr_info("myint is an integer: %d\n", myint);
	pr_info("mylong is a long integer: %ld\n", mylong);
	pr_info("mystring is a string: %s\n", mystring);

	for (i = 0; i < ARRAY_SIZE(myintarray); i++)
		pr_info("myintarray[%d] = %d\n", i, myintarray[i]);

	pr_info("got %d arguments for myintarray.\n", arr_argcs);
	return 0;
}

static void __exit hello_5_exit(void)
{
	pr_info("Goodbye, world 5\n");
}

module_init(hello_5_init);
module_exit(hello_5_exit);
```

C Tab Width: 8 Ln 53, Col 39 INS

4.6 Modules Spanning Multiple Files

Sometimes it makes sense to divide a kernel module between several source files.

Here is an example of such a kernel module.

```c
/*
 * start.c - Illustration of multi filed modules
 */

#include <linux/kernel.h> /* We are doing kernel work */
#include <linux/module.h> /* Specifically, a module */

int init_module(void)
{
    pr_info("Hello, world - this is the kernel speaking\n");
    return 0;
}

MODULE_LICENSE("GPL");
```

The next file:

```c
/*
 * stop.c - Illustration of multi filed modules
 */

#include <linux/kernel.h> /* We are doing kernel work */
#include <linux/module.h> /* Specifically, a module  */

void cleanup_module(void)
{
    pr_info("Short is the life of a kernel module\n");
}

MODULE_LICENSE("GPL");
```

And finally, the makefile:

```
obj-m += hello-1.o
obj-m += hello-2.o
obj-m += hello-3.o
obj-m += hello-4.o
obj-m += hello-5.o
obj-m += startstop.o
startstop-objs := start.o stop.o

PWD := $(CURDIR)

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

This is the complete makefile for all the examples we have seen so far. The first five lines are nothing special, but for the last example we will need two lines. First, we invent an object name for our combined module, second, we tell make what object files are part of that module.

4.7 Building modules for a precompiled kernel

Obviously, we strongly suggest you to recompile your kernel, so that you can enable a number of useful debugging features, such as forced module unloading (MODULE_FORCE_UNLOAD): when this option is enabled, you can force the kernel to unload a module even when it believes it is unsafe, via a sudo rmmod -f module command. This option can save you a lot of time and a number of reboots during the development of a module. If you do not want to recompile your kernel then you should consider running the examples within a test distribution on a virtual machine. If you mess anything up then you can easily reboot or restore the virtual machine (VM).

There are a number of cases in which you may want to load your module into a precompiled running kernel, such as the ones shipped with common Linux distributions, or a kernel you have compiled in the past. In certain circumstances you could require to compile and insert a module into a running kernel which you are not allowed to recompile, or on a machine that you prefer not to reboot. If you can’t think of a case that will force you to use modules for a precompiled kernel you might want to skip this and treat the rest of this chapter as a big footnote.

Now, if you just install a kernel source tree, use it to compile your kernel module and you try to insert your module into the kernel, in most cases you would obtain an error as follows:

insmod: ERROR: could not insert module poet.ko: Invalid module format

Less cryptic information is logged to the systemd journal:

kernel: poet: disagrees about version of symbol module_layout

In other words, your kernel refuses to accept your module because version strings (more precisely, version magic, see include/linux/vermagic.h) do not match. Incidentally, version magic strings are stored in the module object in the form of a static string, starting with vermagic:. Version data are inserted in your module when it is linked against the kernel/module.o file. To inspect version magics and other strings stored in a given module, issue the command modinfo module.ko:

```
$ modinfo hello-4.ko
description:    A sample driver
author:         LKMPG
license:        GPL
srcversion:     B2AA7FBFCC2C39AED665382
depends:
retpoline:      Y
name:           hello_4
vermagic:       5.4.0-70-generic SMP mod_unload modversions
```

To overcome this problem we could resort to the --force-vermagic option, but this solution is potentially unsafe, and unquestionably unacceptable in production modules. Consequently, we want to compile our module in an environment which was identical to the one in which our precompiled kernel was built. How to do this, is the subject of the remainder of this chapter.

First of all, make sure that a kernel source tree is available, having exactly the same version as your current kernel. Then, find the configuration file which was used to compile your precompiled kernel. Usually, this is available in your current boot directory, under a name like config-5.14.x. You may just want to copy it to your kernel source tree: cp /boot/config-`uname -r` .config.

Let's focus again on the previous error message: a closer look at the version magic strings suggests that, even with two configuration files which are exactly the same, a slight difference in the version magic could be possible, and it is sufficient to prevent insertion of the module into the kernel. That slight difference, namely the custom string which appears in the module's version magic and not in the kernel's one, is due to a modification with respect to the original, in the makefile that some distributions include. Then, examine your Makefile, and make sure that the specified version information matches exactly the one used for your current kernel. For example, your makefile could start as follows:

```
VERSION = 5
PATCHLEVEL = 14
SUBLEVEL = 0
EXTRAVERSION = -rc2
```

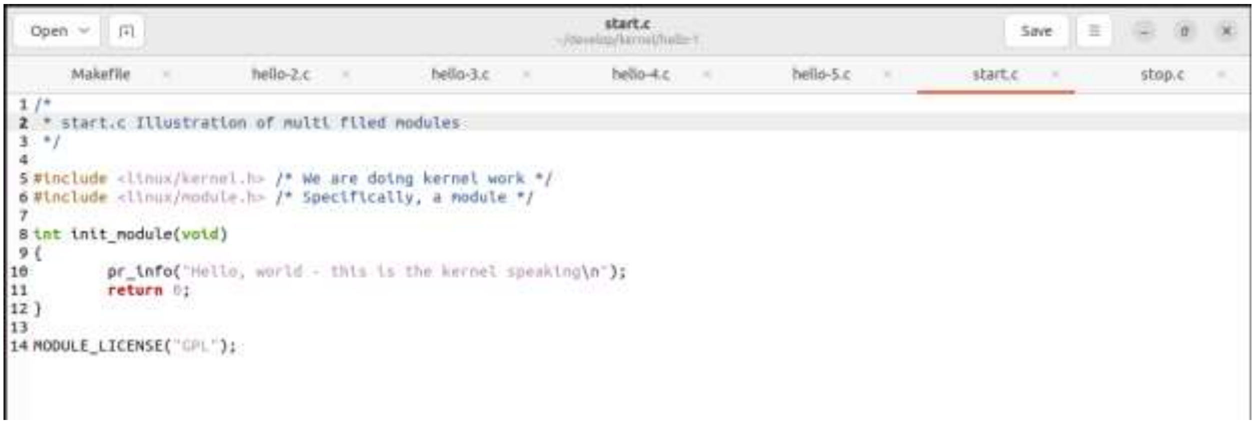
In this case, you need to restore the value of symbol EXTRAVERSION to -rc2. We suggest to keep a backup copy of the makefile used to compile your kernel available in /lib/modules/5.14.0-rc2/build. A simple command as following should suffice.

```
cp /lib/modules/`uname -r`/build/Makefile linux-`uname -r`
```

Here linux-`uname -r` is the Linux kernel source you are attempting to build. Now, please run make to update configuration and version headers and objects:

```
$ make
  SYNC    include/config/auto.conf.cmd
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTCC  scripts/kconfig/util.o
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTLD  scripts/kconfig/conf
```

If you do not desire to actually compile the kernel, you can interrupt the build process (CTRL-C) just after the SPLIT line, because at that time, the files you need are ready. Now you can turn back to the directory of your module and compile it: It will be built exactly according to your current kernel settings, and it will load into it without any errors.



```
/*
 * start.c Illustration of multi filed modules
 */

#include <linux/kernel.h> /* We are doing kernel work */
#include <linux/module.h> /* Specifically, a module */

int init_module(void)
{
        pr_info("Hello, world - this is the kernel speaking\n");
        return 0;
}

MODULE_LICENSE("GPL");
```

```
/*
 * stop.c - Illustration of multi filed modules.
 */

#include <linux/kernel.h> /* We are doing kernel work. */
#include <linux/module.h> /* Specifically, a module. */

void cleanup_module(void)
{
        pr_info("Short is life of a kernel module.\n");
}

MODULE_LICENSE("GPL");
```

```
obj-m += hello-1.o
obj-m += hello-2.o
obj-m += hello-3.o
obj-m += hello-4.o
obj-m += hello-5.o
obj-m += startstop.o

PWD := $(CURDIR)

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
root@Ubuntu:/home/mobi/develop/kernel/hello-1# make
make -C /lib/modules/5.15.0-43-generic/build M=/home/mobi/develop/kernel/hello-1 modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-43-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0
  You are using:           gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0
  CC [M]  /home/mobi/develop/kernel/hello-1/hello-5.o
make[2]: *** No rule to make target '/home/mobi/develop/kernel/hello-1/startstop.o', needed by '/home/mobi/develop/kernel/hello-1/mo
dules.order'.  Stop.
make[1]: *** [Makefile:1875: /home/mobi/develop/kernel/hello-1] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-43-generic'
make: *** [Makefile:11: all] Error 2
root@Ubuntu:/home/mobi/develop/kernel/hello-1#
```

```
root@Ubuntu:/home/mobi/develop/kernel/hello-1# ls
hello-1.c   hello-1.mod    hello-1.mod.o  hello-2.c  hello-3.c  hello-4.c  hello-5.c  Makefile       Module.symvers  stop.c
hello-1.ko  hello-1.mod.c  hello-1.o      hello-2.o  hello-3.o  hello-4.o  hello-5.o  modules.order  start.c
root@Ubuntu:/home/mobi/develop/kernel/hello-1# chmod a+x Module.symvers
root@Ubuntu:/home/mobi/develop/kernel/hello-1# ls
hello-1.c   hello-1.mod    hello-1.mod.o  hello-2.c  hello-3.c  hello-4.c  hello-5.c  Makefile       Module.symvers  stop.c
hello-1.ko  hello-1.mod.c  hello-1.o      hello-2.o  hello-3.o  hello-4.o  hello-5.o  modules.order  start.c
root@Ubuntu:/home/mobi/develop/kernel/hello-1# ./Module.symvers
root@Ubuntu:/home/mobi/develop/kernel/hello-1#
```

```
root@Ubuntu:/home/mobi/develop/kernel/hello-1# ls -a
.               hello-1.mod         hello-1.o         hello-3.c         .hello-4.o.cmd  modules.order          stop.c
..              hello-1.mod.c       .hello-1.o.cmd    hello-3.o         hello-5.c       .modules.order.cmd
hello-1.c       .hello-1.mod.cmd    hello-2.c         .hello-3.o.cmd    hello-5.o       Module.symvers
hello-1.ko      hello-1.mod.o       hello-2.o         hello-4.c         .hello-5.o.cmd  .Module.symvers.cmd
.hello-1.ko.cmd .hello-1.mod.o.cmd  .hello-2.o.cmd    hello-4.o         Makefile        start.c
root@Ubuntu:/home/mobi/develop/kernel/hello-1# ./Module.symvers
root@Ubuntu:/home/mobi/develop/kernel/hello-1# ./modules.order
./modules.order: line 1: /home/mobi/develop/kernel/hello-1/hello-1.ko: Permission denied
root@Ubuntu:/home/mobi/develop/kernel/hello-1# chmod a+x hello-1.ko
root@Ubuntu:/home/mobi/develop/kernel/hello-1# ./modules.order
./modules.order: line 1: /home/mobi/develop/kernel/hello-1/hello-1.ko: cannot execute binary file: Exec format error
root@Ubuntu:/home/mobi/develop/kernel/hello-1#
```

The End.

Special Thanks to Dr. Sharifi for all her efforts.