操作系统 Lab 02

姓名: 雍崔扬
学号: 21307140051

Task 1: 初窥内核模块

生成 Lab 2 Task 1 & 2 的代码框架并编译:

```
LABS=kernel_modules/1-2-test-mod make skels
make build
```

完成后，`make console` 进入虚拟环境就可以在 `skels/kernel_modules` 目录下看到:

```
root@node0:~/linux/tools/labs/skels/kernel_modules# ls -al
total 12
drwxr-xr-x 3 root root 4096 Sep 22 02:24 .
drwxr-xr-x 3 root root 4096 Sep 22 02:24 ..
drwxr-xr-x 2 root root 4096 Sep 22 02:24 1-2-test-mod
```

在虚拟环境中，进入 `/home/root/skels/kernel_modules/1-2-test-mod`，完成以下任务:

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

MODULE_DESCRIPTION("Simple module");
MODULE_AUTHOR("Kernel Hacker");
MODULE_LICENSE("GPL");

static int my_hello_init(void)
{
    pr_debug("Hello!\n");
    return 0;
}

static void hello_exit(void)
{
    pr_debug("Goodbye!\n");
}

module_init(my_hello_init);
module_exit(hello_exit);
```

- ① 加载此内核模块:

  ```
  insmod hello_mod.ko
  ```

- ② 查看已加载的内核模块，确认此模块已加载:

  ```
  lsmod | grep hello_mod
  ```

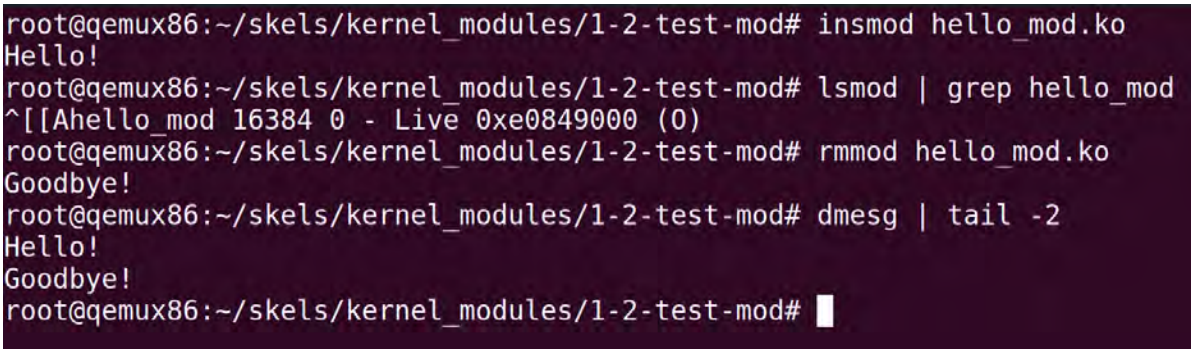
- ③ 卸载此内核模块:

```
rmmod hello_mod.ko
```

- ④ 通过 `dmesg` 命令查看内核记加载/卸载所打印的信息:

```
dmesg | tail -2
```

上述操作的截图:



Task 2: printk

在内核模块开发中，`printk` 可用于在内核态输出日志信息:

```
printk(KERN_INFO "Message: %s\n", arg);
```

其中 `KERN_INFO` 指定了该消息的日志级别.
常用的日志级别如下:

名称	字符串	别名函数
KERN_EMERG	"0"	pr_emerg()
KERN_ALERT	"1"	pr_alert()
KERN_CRIT	"2"	pr_crit()
KERN_ERR	"3"	pr_err()
KERN_WARNING	"4"	pr_warn()
KERN_NOTICE	"5"	pr_notice()
KERN_INFO	"6"	pr_info()
KERN_DEBUG	"7"	pr_debug()

日志级别指定了消息的重要性，数字越低代表重要性越高.
内核会根据消息的日志级别和当前的 `console_loglevel` (一个内核变量) 来决定是否将其打印到控制台 (console) 中
只有级别小于 `console_loglevel` 的消息才会被打印到控制台中.

默认的控制台日志级别可以通过 `/proc/sys/kernel/printk` 配置.
例如执行以下命令:

```
echo 5 > /proc/sys/kernel/printk
```

这将使所有日志级别 < 5 的消息显示在控制台上，
而日志级别 ≥ 5 的 `KERN_NOTICE`，`KERN_INFO`，`KERN_DEBUG` 将不会显示在控制台中.
我们可以使用 `dmesg` 命令查看通过 `printk` 打印但未显示在控制台上的消息.

在 Task 1 中，当我们加载/卸载内核模块时，虚拟环境控制台会打印出 `Hello!` 与 `Goodbye!`
通过查看 `hello_mod.c` 源码可知，这两条消息的日志级别是 `KERN_DEBUG`

现在，我们不希望级别为 `KERN_DEBUG` 的内核消息被打印到控制台中.
请妥善配置虚拟环境，禁止 `KERN_DEBUG` 的内核消息被打印到控制台中.

- ① 查看当前的控制台日志级别:

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

  该命令将显示一个四个数字的输出，代表不同的日志级别，其中第一个数字表示 `console_loglevel` (结果显示为 15)

- ② 设置新的控制台日志级别为 7:

  ```
  echo 7 > /proc/sys/kernel/printk
  ```

- ③ 再次查看当前的控制台日志级别，以确认是否成功修改了 `console_loglevel`:

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

  结果显示成功将 `console_loglevel` 从 15 变为了 7.

- ④ 重试 Task 1 中的指令:

  ```
  insmod hello_mod.ko
  lsmod | grep hello_mod
  rmmod hello_mod.ko
  dmesg | tail -2
  ```

  结果显示成功禁止了 `KERN_DEBUG` 的内核消息被打印到控制台中.

上述操作的截图:

```
root@qemux86:~/skels/kernel_modules/1-2-test-mod# insmod hello_mod.ko
Hello!
root@qemux86:~/skels/kernel_modules/1-2-test-mod# lsmod | grep hello_mod
^[[Ahello_mod 16384 0 - Live 0xe0849000 (O)
root@qemux86:~/skels/kernel_modules/1-2-test-mod# rmmod hello_mod.ko
Goodbye!
root@qemux86:~/skels/kernel_modules/1-2-test-mod# dmesg | tail -2
Hello!
Goodbye!
root@qemux86:~/skels/kernel_modules/1-2-test-mod# cat /proc/sys/kernel/printk
15      4       1       7
root@qemux86:~/skels/kernel_modules/1-2-test-mod# echo 7 > /proc/sys/kernel/prin
tk
root@qemux86:~/skels/kernel_modules/1-2-test-mod# cat /proc/sys/kernel/printk
7       4       1       7
root@qemux86:~/skels/kernel_modules/1-2-test-mod#
root@qemux86:~/skels/kernel_modules/1-2-test-mod# insmod hello_mod.ko
root@qemux86:~/skels/kernel_modules/1-2-test-mod# lsmod | grep hello_mod
hello_mod 16384 0 - Live 0xe0849000 (O)
root@qemux86:~/skels/kernel_modules/1-2-test-mod# rmmod hello_mod.ko
root@qemux86:~/skels/kernel_modules/1-2-test-mod# dmesg | tail -2
Hello!
Goodbye!
root@qemux86:~/skels/kernel_modules/1-2-test-mod#
```

Task 3: 获取进程 PID

生成 Lab 2 Task 3 的代码框架并编译:

```
LABS=kernel_modules/7-list-proc make skels
make build
```

① 完成 `list_proc.c` 中的 `TODO` 内容以显示当前进程的进程的 PID 和可执行文件的名称.
信息必须在加载和卸载模块时显示.

- 在 Linux 内核中，进程由结构体 `task_struct` 描述，
  而当前正在运行的进程的结构体指针由全局变量 `current` (类型为 `struct task_struct*`) 给出
  要使用 `current`，需要包含定义 `struct task_struct` 的头文件，即 `linux/sched.h`
- 为了获取进程的 PID 和可执行文件的名称，需要访问 `task_struct` 的某些字段:
  `current->pid` 用于获取进程的 PID， `current->comm` 用于获取可执行文件的名称.

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
/* TODO: add missing headers */
#include <linux/sched.h>        // For struct task_struct
#include <linux/sched/signal.h> // For for_each_process macro

MODULE_DESCRIPTION("List current processes");
MODULE_AUTHOR("Kernel Hacker");
MODULE_LICENSE("GPL");
雍崔
static int my_proc_init(void)
{
    struct task_struct *p;

    /* TODO: print current process pid and its name */
    printk(KERN_INFO "Current process: PID = %d, Name = %s\n", current->pid,
current->comm);
```

```
    return 0;
}

static void my_proc_exit(void)
{
    /* TODO: print current process pid and name */
    printk(KERN_INFO "Unloading module: Current process: PID = %d, Name = %s\n",
current->pid, current->comm);
}

module_init(my_proc_init);
module_exit(my_proc_exit);
```

② 在虚拟环境中，进入 `/home/root/skels/kernel_modules/7-list-proc`
重复加载/卸载操作，你会发现显示的进程的 PID 不同.
这是因为内核在加载模块时会从可执行文件 `/sbin/insmod` 创建一个新进程，
而在卸载模块时又会从可执行文件 `/sbin/rmmod` 创建另一个新进程.

```
insmod list_proc.ko
lsmod | grep list_proc
rmmod list_proc.ko
```

上述操作的截图:

```
root@qemux86:~# cd /home/root/skels/kernel_modules/7-list-proc
root@qemux86:~/skels/kernel_modules/7-list-proc# insmod list_proc.ko
Current process: PID = 275, Name = insmod
root@qemux86:~/skels/kernel_modules/7-list-proc# lsmod | grep list_proc
list_proc 16384 0 - Live 0xe0831000 (O)
root@qemux86:~/skels/kernel_modules/7-list-proc# rmmod list_proc.ko
Unloading module: Current process: PID = 278, Name = rmmod
root@qemux86:~/skels/kernel_modules/7-list-proc#
```

Task 4: PS

① 在 Task 3 所实现的内核模块的基础上，使其在插入内核模块时显示系统中所有进程的信息，而不仅仅是当前进程的信息.
之后，将获得的结果与 `ps` 命令的输出进行比较.

- 系统中的进程结构是一个循环列表.
  可以采用 `for_each_process(p)` 来访问所有的进程.

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
/* TODO: add missing headers */
#include <linux/sched.h>        // For struct task_struct
#include <linux/sched/signal.h> // For for_each_process macro

MODULE_DESCRIPTION("List current processes");
MODULE_AUTHOR("Kernel Hacker");
MODULE_LICENSE("GPL");

static int my_proc_init(void)
```

```
{
    struct task_struct *p;

    /* TODO: print current process pid and its name */
    printk(KERN_INFO "Current process: PID = %d, Name = %s\n", current->pid,
current->comm);

    /* TODO: print the pid and name of all processes */
    printk(KERN_INFO "Listing all processes:\n");
    for_each_process(p)
    {
        printk(KERN_INFO "Process: PID = %d, Name = %s\n", p->pid, p->comm);
    }

    return 0;
}

static void my_proc_exit(void)
{
    /* TODO: print current process pid and name */
    printk(KERN_INFO "Unloading module: Current process: PID = %d, Name = %s\n",
current->pid, current->comm);
}

module_init(my_proc_init);
module_exit(my_proc_exit);
```

② 在虚拟环境中，进入 `/home/root/skels/kernel_modules/7-list-proc` 加载/卸载 `list_proc.ko`

```
insmod list_proc.ko
lsmod | grep list_proc
rmmod list_proc.ko
```

上述操作的截图:

```
root@qemux86:~# cd /home/root/skels/kernel_modules/7-list-proc
root@qemux86:~/skels/kernel_modules/7-list-proc# insmod list_proc.ko
list_proc: loading out-of-tree module taints kernel.
Current process: PID = 249, Name = insmod
Listing all processes:
Process: PID = 1, Name = init
Process: PID = 2, Name = kthreadd
Process: PID = 3, Name = rcu_gp
Process: PID = 4, Name = rcu_par_gp
Process: PID = 5, Name = kworker/0:0
Process: PID = 6, Name = kworker/0:0H
Process: PID = 7, Name = kworker/u2:0
Process: PID = 8, Name = mm_percpu_wq
Process: PID = 9, Name = ksoftirqd/0
Process: PID = 10, Name = rcu_sched
Process: PID = 11, Name = migration/0
Process: PID = 12, Name = cpuhp/0
Process: PID = 13, Name = kdevtmpfs
Process: PID = 14, Name = netns
Process: PID = 15, Name = oom_reaper
Process: PID = 16, Name = writeback
Process: PID = 38, Name = kblockd
Process: PID = 39, Name = kworker/0:1
Process: PID = 40, Name = kworker/0:1H
Process: PID = 41, Name = kswapd0
Process: PID = 42, Name = cifsiod
Process: PID = 43, Name = smb3decryptd
Process: PID = 44, Name = cifsfileinfoput
Process: PID = 45, Name = cifsoplockd
Process: PID = 47, Name = acpi_thermal_pm
Process: PID = 48, Name = kworker/u2:1
Process: PID = 49, Name = khvcd
Process: PID = 50, Name = kworker/0:2
Process: PID = 51, Name = ipv6_addrconf
Process: PID = 52, Name = kmemleak
Process: PID = 54, Name = cifsd
Process: PID = 94, Name = cifsd
Process: PID = 121, Name = kworker/u2:2
Process: PID = 218, Name = syslogd
Process: PID = 220, Name = klogd
Process: PID = 227, Name = getty
Process: PID = 228, Name = sh
Process: PID = 249, Name = insmod
root@qemux86:~/skels/kernel_modules/7-list-proc# lsmod | grep list_proc
list_proc 16384 0 - Live 0xe0869000 (O)
root@qemux86:~/skels/kernel_modules/7-list-proc# rmmod list_proc.ko
Unloading module: Current process: PID = 252, Name = rmmod
root@qemux86:~/skels/kernel_modules/7-list-proc#
```

③ 与 `ps` 命令对比:

```
root@qemux86:~/skels/kernel_modules/7-list-proc# ps
  PID USER       VSZ STAT COMMAND
    1 root      2004 S    init [5]
    2 root         0 SW   [kthreadd]
    3 root         0 IW<  [rcu_gp]
    4 root         0 IW<  [rcu_par_gp]
    5 root         0 IW   [kworker/0:0-cif]
    6 root         0 IW<  [kworker/0:0H-ev]
    7 root         0 IW   [kworker/u2:0-ev]
    8 root         0 IW<  [mm_percpu_wq]
    9 root         0 SW   [ksoftirqd/0]
   10 root         0 IW   [rcu_sched]
   11 root         0 SW   [migration/0]
   12 root         0 SW   [cpuhp/0]
   13 root         0 SW   [kdevtmpfs]
   14 root         0 IW<  [netns]
   15 root         0 SW   [oom_reaper]
   16 root         0 IW<  [writeback]
   38 root         0 IW<  [kblockd]
   39 root         0 IW   [kworker/0:1]
   40 root         0 IW<  [kworker/0:1H-kb]
   41 root         0 SW   [kswapd0]
   42 root         0 IW<  [cifsiod]
   43 root         0 IW<  [smb3decryptd]
   44 root         0 IW<  [cifsfileinfoput]
   45 root         0 IW<  [cifsoplockd]
   47 root         0 IW<  [acpi_thermal_pm]
   48 root         0 IW   [kworker/u2:1-fl]
   49 root         0 SW   [khvcd]
   50 root         0 IW   [kworker/0:2-cif]
   51 root         0 IW<  [ipv6_addrconf]
   52 root         0 SWN  [kmemleak]
   54 root         0 SW   [cifsd]
   94 root         0 SW   [cifsd]
  121 root         0 IW   [kworker/u2:2-ev]
  218 root      2828 S    /sbin/syslogd -n -O /var/log/messages
  220 root      2828 S    /sbin/klogd -n
  227 root      2828 S    /sbin/getty 38400 tty1
  228 root      2972 S    -sh
  254 root      2976 R    ps
root@qemux86:~/skels/kernel_modules/7-list-proc#
```

内核模块 `list_proc.ko` 和 linux 的 `ps` 命令的输出具有以下区别:

- 内核模块 `list_proc.ko` 的输出只有 `PID` (进程 ID) 和 `Name` (进程名称) 两项;

  ```
  Process: PID = 1, Name = init
  ```

  而 `ps` 命令的输出有 `PID`、`USER` (用户)、`VSZ` (虚拟内存)、`STAT` (状态) 和 `COMMAND` (正在执行的命令)
  其中 `COMMAND` 字段还会输出进程的命令行参数.

  ```
      1 root      2004 S    init [5]
  ```

- 内核模块 `list_proc.ko` 列出的最后一项进程是 `Process: PID = 249, Name = insmod`
  这是系统为装载内核模块的指令 `insmod` 创建的进程，以便在用户空间中执行.
  而 `ps` 列出的最后一项进程是 `254  root 2976  R  ps`
  这表示执行 `ps` 命令时所创建的进程.