Linux kernel

Tutorials to discover how to do some stuff on linux kernel. If you have any suggestions or if you find any problem, please report it !

Important

These tutorials are based on linux kernel 4.x. It is very likely to not work if you're using an older (or newer !) version.

Summary

I. Linux kernel articles

1. Basics

You now have two possibilities:

- You download a linux (ubuntu for example) and you launch it in a virtual machine.
- You download qemu and the linux kernel.

Since there already have a lot of tutorials that explain how to run a linux in a virtual machine, I won't give an explanation. So you can move to another tutorial. However, for the others, I'll explain step by step how to run linux kernel with qemu.

First, download qemu.

Then, we have to build a busybox. Here are the commands you have to enter:

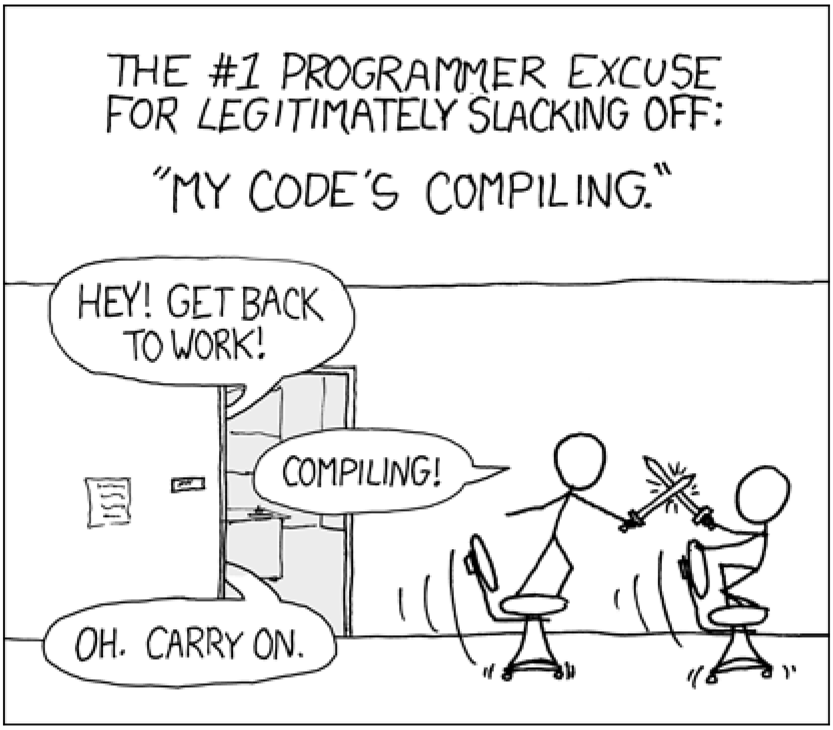
```
cd # to go back to your home directory
mkdir linux_test # now we create a linux folder where everything will be done
cd linux_test
git clone git://git.busybox.net/busybox
make defconfig # All features, no debug
make menuconfig # Setup static linking
make
sudo make install # You can do this without being root but it can fails if you're not
```

Now let's build a rootfs (without it, launching linux kernel will be hard !):

```
dd if=/dev/zero of=disk.img bs=1M count=16
mkfs.ext4 disk.img -L root
mkdir mnt
mount disk.img mnt
cp -r ../busybox/_install/* mnt # the busybox can be different on your computer, just replace it
# Setup mounts
cd mnt
mkdir -p etc/init.d proc sys dev
echo 'proc    /proc   proc    defaults' >> etc/fstab
echo 'sysfs   /sys    sysfs   defaults' >> etc/fstab
echo 'mount -a' >> etc/init.d/rcS
chmod +x etc/init.d/rcS
```

Now let's get to the long part. Don't worry, it's not complicated, just very long. You'll understand why in a few seconds. Now let's download and compile the linux kernel:

```
git clone --depth 1 git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
make defconfig
make
```



And now just wait. Take a coffee or something.

Done? Good! Now we can start!

Launching qemu

Very simple:

```
qemu-system-x86_64 -kernel /path/to/linux/arch/x86/boot/bzImage -hda /path/to/rootfs/disk.img -a
```

If it launched successfully, congratulations! To quit qemu, you now just have to kill it (no kidding). To help you:

```
kill $(ps aux | grep '[q]emu' | awk '{print $2}')
```

Here is ending the first tutorial.

2. Module

We can now start. Here, we'll learn how to create a static module (which is build with the kernel).

First, go to your linux source folder. Then create a folder in it, for example:

```
mkdir my_module
```

Go in it and let's create our first module. Create a file (my_module.c for example) and put this code in:

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
    pr_info("my_module: Hello world\n");
    return 0;
}

static void hello_exit(void)
{
    pr_info("my_module: Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Note the two very important calls at the end:

```
module_init(hello_init);
module_exit(hello_exit);
```

That's how linux knows what function to call. The init function has to return an integer and the exit one has to return nothing. The two functions don't take arguments. I used pr_info to display messages but you can also use printk.

Now create a Makefile next to your file and put in it (of course, if you created a file called 'hello.c', you will have to set 'hello.o':

```
obj-y := my_module.o
```

Now go back to the linux main folder and add your module path to this line:

```
core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/
```

Like this:

```
core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ my_module/
```

Now you just have to build your linux (don't worry, it won't take a while this time):

```
make
```

Launch qemu and you should see the two messages of your module. If you want to see the kernel messages again, just enter the following command:

```
dmesg
```

And if you only want to see your module's messages:

```
dmesg | grep "my_module"
```

Here ends this tutorial !

3. Syscall

Here is a very interesting part of the linux kernel programming! You always wondered "how can I add a syscall in my linux ?"? Here is the answer!

Go to your module folder and create a new one in it:

```
mkdir syscall
```

Good. Now add its path to the current Makefile (the one in linux_test/linux/my_module/):

```
obj-y += syscall/
```

And you should also move your hello.c in a subfolder (it will get dirty very quickly otherwise).

Now go in your syscall folder and create a syscall.c file. Put in it:

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

SYSCALL_DEFINE0(hello)
{
  usleep_range(1000, 100000);
  return 0;
}
```

Very easy isn't it ? But that's not all ! You have to do one more thing (or two, depending on how we look at it...) ! You have to go the file(s) where syscalls are declared. Where ?

```
cd ~/linux_test/linux/arch/x86/syscalls/
# on more recent linux versions:
cd ~/linux_test/linux/arch/x86/entry/syscalls/
```

We are interested here by syscall_32.tbl and syscall_64.tbl. To add a syscall, go to the end of a file and add this line in (the first line is just an example):

```
356 i386    memfd_create        sys_memfd_create
357 i386    hello           sys_hello
```

Be very careful in the syscall_64.tbl file ! If there is a "x32" in the line:

```
544     x32     io_submit               compat_sys_io_submit
```

You're not in the good place. Try to search the end of lines like this:

```
320     common  kexec_file_load         sys_kexec_file_load
```

Now you just have to add your syscall:

```
321     common  hello                   sys_hello
```

You can now build but if you want to test your syscall, just modify or create a new module and put in it:

```
syscall(_NR_hello, 0);
```

If it says `_NR_hello` isn't defined, replace it with your syscall number:

```
syscall(357, 0);
```

Everything's good ? Cool ! Now if you want to add parameter to your syscall:

```
SYSCALL_DEFINE2(hello, unsigned int, begin, unsigned int, end)
{
  usleep_range(begin, end);
  return 0;
}
```

You will notice that SYSCALL_DEFINE0 is now SYSCALL_DEFINE2. I'm sure you already guessed but I say it: the number at the end is the number of parameter that your syscall needs.

Precision

If you want your syscall to take a pointer as parameter, just know that you can't use "just like that". You're in kernel space, so you have to "import" (I simplify a lot) the pointer. Example:

```
SYSCALL_DEFINE3(hello, unsigned int, begin, unsigned int, end, unsigned int*, ptr)
{
  usleep_range(begin, end);
  copy_to_user(ptr, &end, sizeof(ptr));
  return 0;
}
```

You're welcome !

More ?

If you want more information about syscalls, here is a little article very interesting which speak about it.

4. Overwrite syscall

Sounds terrifying, right ? Actually, I found it very cool so that's why I wanted to share it ! Let's begin. Create all the folders and stuff and put in the file:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/mm.h>
#include <linux/mman.h>
#include <linux/syscalls.h>
#include <linux/unistd.h>

unsigned long *syscall_table = NULL;
// corresponds to the syscall that will be replaced
asmlinkage long (*original)(unsigned int size, unsigned long *ptr);
unsigned long original_cr0;

#define SYSCALL_TO_REPLACE _NR_epoll_ctl_old

asmlinkage int new_syscall(unsigned int size, unsigned long *ptr)
{
  printk("I'm the syscall which overwrittes the previous !\n");
  return 0;
}

static unsigned long **find_syscall_table(void)
{
  unsigned long int offset = PAGE_OFFSET;
  unsigned long **sct;

  while (offset < ULLONG_MAX) {
    sct = (unsigned long **)offset;

    if (sct[_NR_close] == (unsigned long *) sys_close)
      return sct;

    offset += sizeof(void *);
  }

  return NULL;
}

static int _init overwrite_init(void)
{
  syscall_table = (void **) find_syscall_table();
  if (syscall_table == NULL) {
    printk(KERN_ERR"net_malloc: Syscall table is not found\n");
    return -1;
  }
  // wrapper for asm part
  original_cr0 = read_cr0();
  write_cr0(original_cr0 & ~0x00010000);
  original = (void *)syscall_table[SYSCALL_TO_REPLACE];

  // we now overwrite the syscall
  syscall_table[SYSCALL_TO_REPLACE] = (unsigned long *)new_syscall;
  write_cr0(original_cr0);
  printk("net_malloc: Patched! syscall number : %d\n", SYSCALL_TO_REPLACE);
  return 0;
}

static void _exit overwrite_exit(void)
{
  // reset overwritten syscall
  if (syscall_table != NULL) {
    original_cr0 = read_cr0();
    write_cr0(original_cr0 & ~0x00010000);

    syscall_table[SYSCALL_TO_REPLACE] = (unsigned long *)original;

    write_cr0(original_cr0);
  }
}

module_init(overwrite_init);
module_exit(oveerwrite_exit);
```

If you want to test it on a very used syscall, try to replace read or write and replace them with this:

```
asmlinkage int new_syscall(unsigned int fd, const char _user *buf, size_t count)
{
    printk("who is calling the big bad write ?\n");
    return (*original)(fd, buf, count);
}
```

You'll see, the result will be quite fun (or not, it's just a point of view...).

5. Add executable in qemu

It can be useful to have a very specific executable in qemu since there is a very few number of them inside. Actually it's quite simple. Compile your program with the --static option. Example:

```
gcc --static file_to_compile.c -o super_executable
```

Now go to your linux folder (where the mnt folder is) and mount the linux image:

```
sudo mount disk.img mnt
```

Then you just have to copy your executable in it:

```
cp super_executable mnt/bin/
```

Next time you'll launch qemu, you will be able to run your super_executable !

6. Create a dynamic module

It can be very useful to know how to build a dynamic module. But be very careful when you use it because a bad code could corrupt your whole system.

Instead of the standard Makefile:

```
obj-y := file.o
```

We now need to do:

```
KERNELDIR ?= /usr/src/linux-headers-3.13.0-39-generic
PWD := $(shell pwd)
default:
$(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
$(MAKE) -C $(KERNELDIR) M=$(PWD) modules clean
```

Run the make command:

```
make
```

To test it:

```
insmod module.ko
```

And to remove the added module:

```
rmmod module
```

Once again, I repeat: be very careful !

7. Sockets

I'm gonna write a tutorial on kernel sockets. A source code of mine is available here and here:

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

#include <linux/kthread.h>
#include <linux/mutex.h>
#include <linux/semaphore.h>

#include <linux/delay.h>
#include <linux/errno.h>
#include <linux/types.h>

#include <linux/netdevice.h>
#include <linux/ip.h>
#include <linux/in.h>

#define _bswap_16(x) ((unsigned short)(_builtin_bswap32(x) >> 16))

MODULE_LICENSE("Dual BSD/GPL");

#define PORT 1111
#define SERVER_ADDR "192.168.37.138"
struct task_struct *kthread = NULL;
struct mutex mutex;
struct socket *sock = NULL;

static char *init_msg = "Init message";
static int  size_init_msg = 13;
static char *exit_msg = "Exit message";
static int  size_exit_msg = 13;

static void convert_from_addr(const char *addr, unsigned char nbs[4])
{
    int i;
    int value;
    int pos;

    for (i = 0, pos = 0; addr[i] && pos < 4; ++i)
    {
        for (value = 0; addr[i] && addr[i] != '.'; ++i) {
            value *= 10;
            value += addr[i] - '0';
        }
        nbs[pos++] = value;
        if (!addr[i])
            return;
    }
}

static int send_msg(unsigned char *buf, int len)
{
    struct msghdr msg = {0};
    struct iovec iov;
    mm_segment_t oldfs;
    int size = 0;

    iov.iov_base = buf;
    iov.iov_len = len;

    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;

    oldfs = get_fs();
    set_fs(KERNEL_DS);
    size = sock_sendmsg(sock,&msg,len);
    set_fs(oldfs);
    if (size < 0)
        printk("thug_write_to_socket: sock_sendmsg error: %d\n", size);

    return size;
}

static int my_connect(struct socket **s, const char *s_addr)
{
    struct sockaddr_in address = {0};
    int len = sizeof(struct sockaddr_in), ret;
    unsigned char i_addr[4] = {0};
    if ((ret = sock_create(AF_INET, SOCK_STREAM, IPPROTO_TCP, s)) < 0)
    {
        printk("echo_server: sock_create error");
        return ret;
    }
    convert_from_addr(s_addr, i_addr);
    address.sin_addr.s_addr = *(unsigned int*)i_addr;
    address.sin_port = htons(PORT);
    address.sin_family = AF_INET;
    if ((ret = (*s)->ops->connect(sock, (struct sockaddr*)&address, len, 0)) < 0)
    {
        sock_release(*s);
        *s = NULL;
    }
    return ret;
}

static int echo_server_init(void)
{
    int ret;

    if ((ret = my_connect(&sock, "127.0.0.1")) < 0)
    {
        printk("echo_server: connect error");
        return ret;
    }
    printk(KERN_ALERT "echo-server is on !\n");
    send_msg(init_msg, size_init_msg);
    return 0;
}

static void echo_server_exit(void)
{
    send_msg(exit_msg, size_exit_msg);
    if (sock != NULL)
    {
        sock_release(sock);
        sock = NULL;
    }
    printk(KERN_ALERT "echo-server is now ending\n");
}

module_init(echo_server_init);

module_exit(echo_server_exit);
```

8. Send patch to Linux

You made a patch to add a new feature or to solve an issue ? Nice ! But it'd be better to send it to the Linux foundation now ! I'll explain you how in this chapter.

Setup

First, we need to add the missing tools:

```
> apt-get update
> apt-get install git git-email gitk
```

Then you'll need to configure your git if you haven't done it yet:

```
git config --add user.name "Guillaume Gomez"
git config --add user.email "guillaume1.gomez@gmail.com"
```

Once this done, we need to add the configuration for git-email. Add the following lines to your `~/.gitconfig` file:

```
[sendemail]
    smtpencryption = tls
    stmpserver = smtp.gmail.com
    smtpuser = guillaume1.gomez@gmail.com
    smtpserverport = 587
    from = Guillaume Gomez <guillaume1.gomez@gmail.com>
    suppresscc = self
```

Of course, change the information to make them match your own configuration. Please note that some smtp servers might not accept very well git-email so you'll need to do some additional change. For example, on gmail, I had to go to my account and set "Authorize less secured applications" parameter to true.

Once you have done this, we can continue !

Commit messages

The commit messages have to be very explicit to make the work of linux developers easier. For example, if you worked on tty and solved an issue, your commit message should look like:

```
tty: solved [description of issue]
```

Creating the patch file

Once you have commited your changes, you'll need to create the patch file to send. It's very easy to do:

```
> git format-patch -s -n master..fix_branch
```

Where `fix_branch` is the branch where you made the changes (never modify master directly!). You should now have a file called `000x-something.patch`.

Checking patch

Before sending the patch, it would be nice to check if there is any error in it, right? A script does it for you:

```
> scripts/checkpatch.pl 000x-something.patch
```

The output should look like this:

```
total: 0 errors, 0 warnings, 14 lines checked
000x-something.patch has no obvious style problems and is ready for submission.
```

If you have warnings, it's *generally* OK, but please check if there is nothing serious before going forward.

Know whom you should send the patch to

You can't just send your patch to anyone! Once again, a script is provided to get the concerned people list:

```
> scripts/get_maintainer.pl 000x-something.patch
```

And so you should get something like this:

```
Greg Kroah-Hartman <gregkh@linuxfoundation.org> (supporter:TTY LAYER)
Jiri Slaby <jslaby@suse.com> (supporter:TTY LAYER)
linux-kernel@vger.kernel.org (open list)
```

Sending email

Before going any further, please test your patch!

Done?

Good! Before sending the mail, it would be nice to test if everything is working fine, right? Let's send an email to ourselves!

```
> git send-email --to "you@you.com" 000x-something.patch
```

If you have some issue with Perl, please install the missing libraries/packages through a Perl package manager.

If you received the mail, congrats! It means you're now ready to send it to the Linux foundation:

```
> git send-email --cc "you@you.com" --cc "linux-kernel@vger.kernel.org" --to "gregkh@linuxfounda
```

Once this done, you can follow your patch status on mailing list. There are a few websites where you can see them. I usually go on marc.info, but there is also Linux Kernel Mailing Lists and others... I'll let you check. Hope it helped!

8. Going a bit further

What about hosting your memory on another computer? Here's an incomplete implementation (just read, not write):

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

#include <linux/netdevice.h>
#include <linux/ip.h>
#include <linux/in.h>

#include <linux/mm.h>
#include <linux/mman.h>
#include <linux/syscalls.h>
#include <linux/unistd.h>

static char *ip = "127.0.0.1";
module_param(ip, charp, 0);

static int port = 4242;
module_param(port, int, 0);

struct socket *sock = 0;
static unsigned long page = 0;

unsigned long *syscall_table = (unsigned long *)0xffffffff8180ae20;
asmlinkage long (*original)(unsigned int size, unsigned long *ptr);
unsigned long original_cr0;

unsigned long begin_pointer = 0;

#define SYSCALL_TO_REPLACE _NR_epoll_ctl_old

/* NETWORK */
#define READ_ID 1
#define WRITE_ID 2
#define ALLOC_ID 3

static void convert_from_addr(const char *addr, unsigned char nbs[4])
{
    int i;
    int value;
    int pos;

    for (i = 0, pos = 0; addr[i] && pos < 4; ++i)
    {
        for (value = 0; addr[i] && addr[i] != '.'; ++i) {
            value *= 10;
            value += addr[i] - '0';
        }
        nbs[pos++] = value;
        if (!addr[i])
            return;
    }
}

static int get_data(void *buf, int len, int send)
{
    struct msghdr msg = {0};
    struct iovec iov;
    mm_segment_t oldfs;
    int size = 0;

    iov.iov_base = buf;
    iov.iov_len = len;

    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;

    oldfs = get_fs();
    set_fs(KERNEL_DS);
    if (send == 1)
        size = sock_sendmsg(sock, &msg, len);
    else
        size = sock_recvmsg(sock, &msg, len, MSG_WAITALL);
    set_fs(oldfs);
    if (size < 0)
        printk("%s: error: %d\n", send ? "sock_sendmsg" : "sock_recvmsg", size);

    return size;
}

static int send_msg(void *buf, int len)
{
    return get_data(buf, len, 1);
}

static int recv_msg(void *buf, int len)
{
    return get_data(buf, len, 0);
}

static int my_connect(struct socket **s, const char *s_addr)
{
    struct sockaddr_in address = {0};
    int len = sizeof(struct sockaddr_in), ret;
    unsigned char i_addr[4] = {0};
    if ((ret = sock_create(AF_INET, SOCK_STREAM, IPPROTO_TCP, s)) < 0)
    {
        printk("echo_server: sock_create error");
        return ret;
    }
    convert_from_addr(s_addr, i_addr);
    address.sin_addr.s_addr = *(unsigned int*)i_addr;
    address.sin_port = htons(port);
    address.sin_family = AF_INET;
    if ((ret = (*s)->ops->connect(sock, (struct sockaddr*)&address, len, 0)) < 0)
    {
        sock_release(*s);
        *s = NULL;
    }
    return ret;
}

/* NETMALLOC */
static int my_fault(struct vm_area_struct *vma, struct vm_fault *vmf)
{
    int page_size = PAGE_SIZE;
    pr_info("There is page fault !\n");
    page = get_zeroed_page(GFP_KERNEL);
    //Fill page with data get from network
    if (sock != NULL)
    {
        char id = READ_ID;
        send_msg(&id, sizeof(id));
        pr_info("READ id\n");
        send_msg(&begin_pointer, sizeof(begin_pointer)); // begin pointer
        pr_info("page\n");
        send_msg(&vmf->virtual_address, sizeof(unsigned long)); // from pointer
        pr_info("page size \n");
        send_msg((void*)&page_size, sizeof(unsigned int));

        recv_msg((void*)page, PAGE_SIZE);
    }
    vmf->page = virt_to_page(page);
    return 0;
}

struct vm_operations_struct my_vm_ops = {
  .fault = my_fault
};

asmlinkage int new_netmalloc(unsigned int size, unsigned long *ptr)
{
    struct vm_area_struct *vm_area;

    printk("=============> overwrite syscall +  new vm allocation\n");
    begin_pointer = vm_mmap(0, 0, size, PROT_READ | PROT_WRITE | PROT_EXEC,
                MAP_ANONYMOUS | MAP_PRIVATE, 0);
    vm_area = find_vma(current->mm, begin_pointer);
    vm_area->vm_ops = &my_vm_ops;
    copy_to_user(ptr, &begin_pointer, sizeof(begin_pointer));

    if (sock != NULL && size != 0)
    {
        char id = ALLOC_ID;

        send_msg(&id, sizeof(id));
        send_msg(&begin_pointer, sizeof(begin_pointer));
        unsigned int size_page = (size / PAGE_SIZE) * PAGE_SIZE + PAGE_SIZE;
        send_msg((char*)&size_page, sizeof(size));
    }
    return 0;
}

/* Module */

static unsigned long **find_syscall_table(void)
{
    unsigned long int offset = PAGE_OFFSET;
    unsigned long **sct;

    while (offset < ULLONG_MAX) {
        sct = (unsigned long **)offset;

        if (sct[_NR_close] == (unsigned long *) sys_close)
            return sct;

        offset += sizeof(void *);
    }

    return NULL;
}

static int _init mymodule_init(void)
{
    int ret;

    syscall_table = (void **) find_syscall_table();
    if (syscall_table == NULL)
    {
        printk(KERN_ERR"net_malloc: Syscall table is not found\n");
        return -1;
    }
    original_cr0 = read_cr0();

    write_cr0(original_cr0 & ~0x00010000);
    original = (void *)syscall_table[SYSCALL_TO_REPLACE];
    syscall_table[SYSCALL_TO_REPLACE] = (unsigned long *)new_netmalloc;
    write_cr0(original_cr0);
    printk("net_malloc: Patched! netmalloc number : %d\n", SYSCALL_TO_REPLACE);
    if ((ret = my_connect(&sock, "127.0.0.1")) < 0)
    {
        printk("netmalloc: connect error\n");
        return -ENOMEM;
    }
    return 0;
}

static void _exit mymodule_exit(void)
{
    if (sock != NULL)
    {
        sock_release(sock);
        sock = NULL;
    }
    printk("module exit\n");

    // reset syscall
    if (syscall_table != NULL)
    {
        original_cr0 = read_cr0();
        write_cr0(original_cr0 & ~0x00010000);
        syscall_table[SYSCALL_TO_REPLACE] = (unsigned long *)original;
        write_cr0(original_cr0);
    }
}

module_init(mymodule_init);
module_exit(mymodule_exit);
```