> Name: Liqin Zhang, Siwei Ye
>
> ID: 517370910123, 517370910122

VE482 Lab7 Linux Kernel

1. What is a kernel module, and how does it different from a regular library?

A kernel module is a piece of kernel code that can be loaded or unloaded on demand to extend the functionality of the kernel without rebooting the system.[1] Microsoft Windows and UNIX-like systems support this feature, but it has different names in different operating systems, such as FreeBSD called kernel loadable module (abbreviated as kld), Mac OS X called kernel extensions (kernel), and so on. extension (abbreviated as kext). It is also known as Kernel Loadable Modules (KLM), or Kernel Modules (KMOD).

The main differences between the kernel module and a regular library is:

- The kernel module allows the operating system to load dynamically when new features are needed, reducing development and usage difficulties.
- If a kernel module is buggy, it risks damaging the entire system kernel.

2. How to compile a kernel module?

We need a special makefile and the permission of superuser. (src contained in ex2 folder)

- First we create a makefile as follows

```
KERNELDIR=/lib/modules/`uname -r`/build
#ARCH=i386
#KERNELDIR=/usr/src/kernels/`uname -r`-i686

MODULES = helloWorld.ko kernelRead.ko
obj-m += helloWorld.o kernelRead.o

all:
	make -C $(KERNELDIR) M=$(PWD) modules

clean:
	make -C $(KERNELDIR) M=$(PWD) clean

install:
	make -C $(KERNELDIR) M=$(PWD) modules_install

quickInstall:
	cp $(MODULES) /lib/modules/`uname -r`/extra
```

- Then we compile the modules by running `make` in the directory where the modules reside. If the compilation succeeds, the resulting module code is contained in the file `<module_name>.ko`
- Next, we can insert module into the running kernel by typing `insmod <module_name>.ko`
  Any output generated by the `printk`-function will appear in the file /var/log/messages . The best command to watch this output is
  `tail -f /var/log/messages`

- The main kernel module is written as follow.

```c
/*  hello.c - The simplest kernel module.
 */

#include <linux/module.h>  /* Needed by all modules */
#include <linux/kernel.h>  /* Needed for KERN_ALERT */



MODULE_AUTHOR ("Eike Ritter <E.Ritter@cs.bham.ac.uk>");
MODULE_DESCRIPTION ("The standard Hello World Program") ;
MODULE_LICENSE("GPL");

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

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


void cleanup_module(void)
{
  printk(KERN_INFO "Goodbye world 1.\n");
}
```

3. The detailed testing is given in `README.md`.

4. How are `mutex` defined and used? How good is this approach?

`mutex` is defined in `base.c`:

```c
static DEFINE_MUTEX(dadfs_sb_lock);
static DEFINE_MUTEX(dadfs_inodes_mgmt_lock);
```

As is shown in the code above, there are two mutexes: `dadfs_sb_lock` and `dadfs_inodes_mgmt_lock`. Take `dadfs_sb_lock` as an example, to lock the mutex, use

```c
mutex_lock_interruptible(&dadfs_sb_lock)
```

and unlock the mutex with

```c
mutex_unlock(&dadfs_sb_lock)
```

Since the mutexes are global variables, the approach supports global mutex operations. What's more, when using `mutex_lock_interruptible` , the mutex can be interrupted by outer signals, which can avoid some issues in a filesystem.

5. How is information shared between the kernel and user spaces?

In the original source code, information is shared using `copy_to_user` and `copy_from_user`.

In our latest source code, it is shared using `copy_to_iter` and `copy_from_iter` , which are defined in `linux/uio.h`.

6. Document your changes in the README file.

See `README.md`.

---

1. https://wiki.archlinux.org/index.php/Kernel_module_(%E7%AE%80%E4%BD%93%E4%B8%AD%E6%96%87) ↩