VE482 Lab Report

Lab 7 - Fall 2020

Wu Qinhang (518370910041)
Sun Zhimin (518370910219)

Table of Contents

- Linux kernel source code
- clean software updates

Kernel Module

What is a kernel module, and how does it different from a regular library? [1] [2] [3]

- A kernel module is an object file aimed at extending the kernel functionality when it is loaded onto the kernel. It can be treaded as a plugin to the kernel program. A kernel module is linked only to the kernel, and it can be dynamically loaded. In this way, when a new kernel module is introduced, the kernel doesn't need to be re-compiled. However, a regular library is just a collection of relevant functionality that supports different application. If a library of kernel is updated, the kernel needs to be re-compiled.

How to compile a kernel module?

- The module should not be linked to libraries, and the module should be compiled with `kbuild`.

The following example is taken from [1]:

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

MODULE_DESCRIPTION("My kernel module");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

static int dummy_init(void)
{
  pr_debug("Hi\n");
  return 0;
}

static void dummy_exit(void)
{
  pr_debug("Bye\n");
}

module_init(dummy_init);
module_exit(dummy_exit);
```

```makefile
# Makefile
KDIR = /lib/modules/`uname -r`/build

kbuild:
  make -C $(KDIR) M=`pwd`

clean:
  make -C $(KDIR) M=`pwd` clean
```

Kbuild

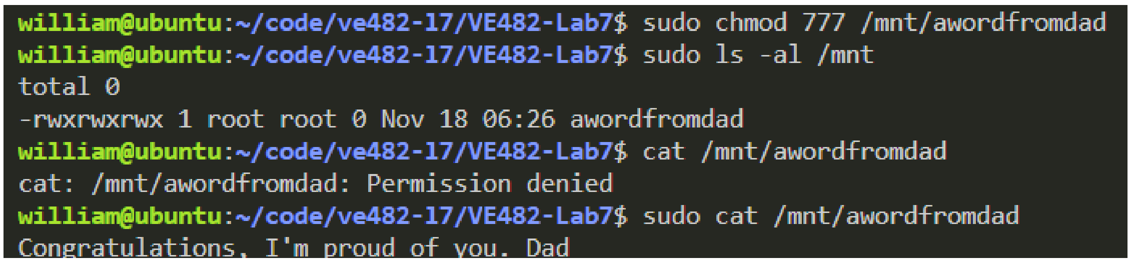
```
EXTRA_CFLAGS = -Wall -g
obj-m        = helloworld.o
```

dadfs

For source code, see `./dadfs/*`.

Build command:

```bash
#!/bin/bash
# check the magic version
uname -r
# check the loop status
lsblk --fs
make
dd bs=4096 count=100 if=/dev/zero of=disk
dd bs=1M count=10 if=/dev/zero of=journal
mke2fs -b 4096 -O journal_dev journal
./mkfs-dadfs disk
# insert the module
sudo insmod ./dadfs.ko
test_journal_dev=$(sudo losetup -f --show journal)
sudo mount -o loop,owner,group,users,journal_path="$test_journal_dev" -t
dadfs disk /mnt
```



```
william@ubuntu:~/code/ve482-17/VE482-Lab7$ sudo chmod 777 /mnt/awordfromdad
william@ubuntu:~/code/ve482-17/VE482-Lab7$ sudo ls -al /mnt
total 0
-rwxrwxrwx 1 root root 0 Nov 18 06:26 awordfromdad
william@ubuntu:~/code/ve482-17/VE482-Lab7$ cat /mnt/awordfromdad
cat: /mnt/awordfromdad: Permission denied
william@ubuntu:~/code/ve482-17/VE482-Lab7$ sudo cat /mnt/awordfromdad
Congratulations, I'm proud of you, Dad
```

Figure 1. "I'm proud of you, Dad"

mutex

How are mutex defined and used? How good is this approach?

Three mutex in total are defined, they are:

- `dadfs_sb_lock` Used for locking super block when action is taken on block
- `dadfs_inodes_mgmt_lock` Used for locking super block when block inode need to be changed
- `dadfs_directory_children_update_lock` Used for locking super block when children of the block is changed

Advantages:

1. Naming of variables are pretty clear.
2. Function can be accomplished with one mutex, but it separates into three parts in order to improve the efficiency.
3. Of course dad your code is perfect! No one knows mutex BETTER then you!

information share

How is information shared between the kernel and user spaces?

First, we need to let kernel allocate the memory, then allow the userspace process to map it into its address space with `mmap()`.

Then, take C as an example, the kernel thread and the userspace process will use different pointers to refer to the memory region:

- Kernel thread will use a pointer to the memory allocated within the kernel address space,
- User space process will use a pointer to the memory region returned by `mmap()`.

changes

```
11c11
< #include <linux/slab.h>
---
> #include <linux/slab.h>
17,19d16
< #if LINUX_VERSION_CODE >= KERNEL_VERSION(3, 11, 0)
< #include <linux/uio.h>
< #endif
358d354
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
361,363d356
```

```
< #else
< ssize_t dadfs_write(struct kiocb * kiocb, struct iov_iter * iov_iter)
< #endif
373d365
<       // struct *i_mapping;
379c371
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
---
>
381,383d372
< #else
<       sb = kiocb->ki_filp->f_inode->i_sb;
< #endif
389,390d377
<
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
392,396d378
<       // int generic_write_checks(struct file *file, loff_t *pos, size_t *count, int isblk);
< #else
<       retval = generic_write_checks(kiocb, iov_iter);
<       // extern ssize_t generic_write_checks(struct kiocb *, struct iov_iter *);
< #endif
399c381
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
---
>
401,404d382
< #else
<       inode = kiocb->ki_filp->f_inode;
< #endif
<
406c384
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
---
>
409,412d386
< #else
<       bh = sb_bread(kiocb->ki_filp->f_inode->i_sb,
<                                                   sfs_inode->data_block_number);
< #endif
422d395
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
424,426d396
< #else
<       buffer += kiocb->ki_pos;
< #endif
434,440c404,406
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
<       if (copy_from_user(buffer, buf, len)) { // (void *to, const void __user *from, __kernel_size_t n);
< #else
<       if (copy_from_iter(buffer, iov_iter->count, iov_iter) == 0) { // FIXME
<               // size_t copy_from_iter(void *addr, size_t bytes, struct iov_iter *i)
< #endif
<               brelse(bh);
---
>
>       if (copy_from_user(buffer, buf, len)) {
>               brelse(bh);
445d410
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
447,449d411
< #else
<       kiocb->ki_pos += iov_iter->count; // FIXME: len
< #endif
477,478d438
<
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
480,482d439
< #else
<       sfs_inode->file_size = (kiocb->ki_pos);
< #endif
485d441
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
488a445
>
490,497d446
< #else
<               ;
<               mutex_unlock(&dadfs_inodes_mgmt_lock);
<               return retval;
<       }
<       mutex_unlock(&dadfs_inodes_mgmt_lock);
<       return iov_iter->count; // TODO: change len
< #endif
502d450
< #if LINUX_VERSION_CODE < KERNEL_VERSION(4, 0, 0)
504,506d451
< #else
<       .write_iter = dadfs_write,
< #endif
664,666d608
<
< // dadfs_bmap
< // TODO: address_space_operations dadfs_a_ops = {}
```

Reference

1. "Kernel modules — The Linux Kernel documentation," *Github.io* , 2020. https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules.html#:~:text=A%20kernel%20module%20(or%20loadable,the%20form%20of%20kernel%20modules . (accessed Nov. 13, 2020). ↩ ↩

2. jetru, "Difference between a module, library and a framework," *Stack Overflow* , Nov. 04, 2010. https://stackoverflow.com/questions/4099975/difference-between-a-module-library-and-a-framework (accessed Nov. 13, 2020). ↩

3. *Oracle.com* , 2010. https://docs.oracle.com/cd/E19253-01/817-5789/emjjr/index.html (accessed Nov. 14, 2020). ↩