VE482 Lab 7 Report

Lan Wang 519370910084

Kaiwen Zhang 519370910188

Dadfs

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

What is a kernel module?[2] "Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. "

- Kernel module works in the kernel space while typically regular library runs in the user space.
- Kernel module is "only linked to the kernel" and uses "different header files".
- "Kernel modules can be dynamically loaded" while typically regular library requires you to recompile to test the changes.
- Kernel modules "must be preemptable & can be interrupted".

How to compile a kernel module?[3]

Two ways:

1. Write our own makefile which "care much about kernel versions".
2. Use `kbuild`.

Write and test all the commands that are only hinted in the README file.

1. compile: `make`

```
lanwang@ubuntu  ~/Desktop/VE482/l7/VE482_lab/l7  main ±  make
make -C /lib/modules/5.11.0-40-generic/build M=/home/lanwang/Desktop/VE482/l7/VE
482_lab/l7 modules
make[1]: Entering directory '/usr/src/linux-headers-5.11.0-40-generic'
  CC [M]  /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/base.o
  LD [M]  /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/dadfs.o
  MODPOST /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/Module.symvers
  CC [M]  /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/dadfs.mod.o
  LD [M]  /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/dadfs.ko
  BTF [M] /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/dadfs.ko
Skipping BTF generation for /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/dadfs.ko
 due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.11.0-40-generic'
cc     mkfs-dadfs.c   -o mkfs-dadfs
```

2. create a small virtual disk (to be formated in dadfs): `dd bs=4096 count=100 if=/dev/zero of=disk`

```
lanwang@ubuntu  ~/Desktop/VE482/l7/VE482_lab/l7  main ±  dd bs=4096 count
=100 if=/dev/zero of=disk
100+0 records in
100+0 records out
409600 bytes (410 kB, 400 KiB) copied, 0.000481692 s, 850 MB/s
```

3. create a small virtual disk (to be used as dadfs' journal): `dd bs=1M count=10 if=/dev/zero of=journal`

```
lanwang@ubuntu  ~/Desktop/VE482/l7/VE482_lab/l7  main ±  dd bs=1M count=1
0 if=/dev/zero of=journal
10+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.0066824 s, 1.6 GB/s
```

4. initialise the journal: `mke2fs -b 4096 -O journal_dev journal`

```
lanwang@ubuntu  ~/Desktop/VE482/l7/VE482_lab/l7  main ±  mke2fs -b 4096 -O
 journal_dev journal
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 2560 4k blocks and 0 inodes
Filesystem UUID: 08400c46-430b-432a-a65c-6b6e924549a3
Superblock backups stored on blocks:

Zeroing journal device:
```

5. format the disk: `./mkfs-dadfs disk`

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# ./mkfs-dadfs disk
Super block written succesfully
root directory inode written succesfully
journal inode written succesfully
welcomefile inode written succesfully
inode store padding bytes (after the three inodes) written sucessfully
Journal written successfully
root directory datablocks (name+inode_no pair for welcomefile) written succesful
ly
padding after the rootdirectory children written succesfully
block has been written succesfully
```

6. load dadfs module: `insmod dadfs.ko`

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# insmod dadfs.ko
```

7. mount disk: `losetup`, `mount` (loop,journal_path)

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# losetup --find --show j
ournal
/dev/loop17
```

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# chmod -R 755 test/
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# mount -o loop,journal_p
ath=/dev/loop17 -t dadfs disk /home/lanwang/Desktop/VE482/l7/VE482_lab/l7/test
```

8. play with dad filesystem: `mkdir`, `mv`, `cp`, `cat`, `rm`, `ls`, `cd`, `touch`, etc.

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# cd test
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7/test# ls
awordfromdad
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7/test# cat awordfromdad
Congratulations, I'm proud of you. Dad
```

9. check the logs: /var/log, `dmesg`

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7/test# dmesg
[    0.000000] Linux version 5.11.0-40-generic (buildd@lgw01-amd64-010) (gcc (Ub
untu 9.3.0-17ubuntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #44~20
.04.2-Ubuntu SMP Tue Oct 26 18:07:44 UTC 2021 (Ubuntu 5.11.0-40.44~20.04.2-gener
ic 5.11.22)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.11.0-40-generic root=UUI
D=6e8ddfdf-8151-47ab-b982-a902550cf458 ro find_preseed=/preseed.cfg auto nopromp
t priority=critical locale=en_US quiet
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] Disabled fast string operations
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point regi
```

10. umount disk: `losetup`, `umount`

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7/test# losetup -d /dev/l
oop17
```

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7/test# cd ..
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# umount test
```

11. unload module: `rmmod`

```
root@ubuntu:/home/lanwang/Desktop/VE482/l7/VE482_lab/l7# rmmod dadfs.ko
```

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

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

Listed in the code blocks at the top are the three static mutexs defined by the writter.

And these three mutexs are somehow in a hireachy to ensure different critical regions won't be in race condition.

How good:

- He uses `static` keywords instead of `global` which is strongly recommended when writing kernel modules.
- He uses `mutex_lock_interruptible` instead of `mutex_lock` which ensures the lock can be interrupted in hardware level.
- The hireachy system of mutexs promises the protection to be flexible.

How is information shared between the kernel and user spaces?

Previous: `copy_from_user` & `copy_to_user`

Latest: we have `iov_iter`, so `copy_from_iter` & `copy_to_iter`

Document your changes.

We use `diff  ./dadfs/base.c ./VE482_lab/l7/base.c  --suppress-common-lines --ignore-all-space > output.txt`.

```
17a18,21
> #if LINUX_VERSION_CODE >= KERNEL_VERSION(3, 11, 0)
> #include <linux/uio.h>
> #endif
>
40,41c44
< void dadfs_sb_sync(struct super_block *vsb)
< {
---
> void dadfs_sb_sync(struct super_block *vsb) {
56,57c59
<       struct dadfs_inode *search)
< {
---
>                                         struct dadfs_inode *search) {
```

```
72,73c74
< void dadfs_inode_add(struct super_block *vsb, struct dadfs_inode *inode)
< {
---
> void dadfs_inode_add(struct super_block *vsb, struct dadfs_inode *inode) {
115,116c116
< int dadfs_sb_get_a_freeblock(struct super_block *vsb, uint64_t * out)
< {
---
> int dadfs_sb_get_a_freeblock(struct super_block *vsb, uint64_t *out) {
136c136,137
<       printk(KERN_ERR "No more free blocks available");
---
>         printk(KERN_ERR
>         "No more free blocks available");
154,155c155
<                     uint64_t * out)
< {
---
>                                         uint64_t *out) {
170a171
>
232,233c233
<                     uint64_t inode_no)
< {
---
>                                       uint64_t inode_no) {
271,272c271,276
< ssize_t dadfs_read(struct file * filp, char __user * buf, size_t len,
<             loff_t * ppos)
---
> ssize_t dadfs_read(struct file *filp, char __user
>
> * buf,
> size_t len,
>         loff_t
> * ppos)
293c297,298
<       printk(KERN_ERR "Reading the block number [%llu] failed.",
---
> printk(KERN_ERR
> "Reading the block number [%llu] failed.",
301c306,308
<   if (copy_to_user(buf, buffer, nbytes)) {
---
> if (
> copy_to_user(buf, buffer, nbytes
> )) {
305c312,313
<       return -EFAULT;
---
> return -
> EFAULT;
310c318,319
<   *ppos += nbytes;
---
> *ppos +=
> nbytes;
```

```
312c321,322
<   return nbytes;
---
> return
> nbytes;
316,317c326
< int dadfs_inode_save(struct super_block *sb, struct dadfs_inode *sfs_inode)
< {
---
> int dadfs_inode_save(struct super_block *sb, struct dadfs_inode *sfs_inode) {
335c344,345
<       printk(KERN_INFO "The inode updated\n");
---
>         printk(KERN_INFO
>         "The inode updated\n");
354a365
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
356a368,371
> #else
>
> ssize_t dadfs_write(struct kiocb *kiocb, struct iov_iter *iov_iter)
> #endif
371a387
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
372a389,391
> #else
>     sb = kiocb->ki_filp->f_inode->i_sb;
> #endif
377a397
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
386a407,417
> #else
>     retval = generic_write_checks(kiocb, iov_iter);
>     if (retval)
>         return retval;
>
>     inode = kiocb->ki_filp->f_inode;
>     sfs_inode = DADFS_INODE(inode);
>
>     bh = sb_bread(kiocb->ki_filp->f_inode->i_sb,
>                   sfs_inode->data_block_number);
> #endif
389c420,421
<       printk(KERN_ERR "Reading the block number [%llu] failed.",
---
>         printk(KERN_ERR
>         "Reading the block number [%llu] failed.",
395a428
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
396a430,432
> #else
>     buffer += kiocb->ki_pos;
> #endif
405c441,446
<   if (copy_from_user(buffer, buf, len)) {
---
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
>     if (copy_from_user(buffer, buf, len))
```

```
> #else
>     if (copy_from_iter(buffer, iov_iter->count, iov_iter) == 0)
> #endif
>     {
410a452,453
>
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
411a455,457
> #else
>     kiocb->ki_pos += iov_iter->count; // FIXME: len
> #endif
438a485
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
446a494,503
> #else
>     sfs_inode->file_size = (kiocb->ki_pos);
>     retval = dadfs_inode_save(sb, sfs_inode);
>     if (retval) {
>         mutex_unlock(&dadfs_inodes_mgmt_lock);
>         return retval;
>     }
>     mutex_unlock(&dadfs_inodes_mgmt_lock);
>     return iov_iter->count;
> #endif
450a508
> #if LINUX_VERSION_CODE < KERNEL_VERSION(3, 11, 0)
451a510,512
> #else
>         .write_iter = dadfs_write,
> #endif
479,480c540
<                     umode_t mode)
< {
---
>                                     umode_t mode) {
534c594,595
<       printk(KERN_INFO "New directory creation request\n");
---
>         printk(KERN_INFO
>         "New directory creation request\n");
538c599,600
<       printk(KERN_INFO "New file creation request\n");
---
>         printk(KERN_INFO
>         "New file creation request\n");
552c614,615
<       printk(KERN_ERR "dadfs could not get a freeblock");
---
>         printk(KERN_ERR
>         "dadfs could not get a freeblock");
603,604c666
<             umode_t mode)
< {
---
>                         umode_t mode) {
611,612c673
<              umode_t mode, bool excl)
< {
```

```
---
>                          umode_t mode, bool excl) {
616,617c677
< static struct inode *dadfs_iget(struct super_block *sb, int ino)
< {
---
> static struct inode *dadfs_iget(struct super_block *sb, int ino) {
646,647c706
<                  struct dentry *child_dentry, unsigned int flags)
< {
---
>                              struct dentry *child_dentry, unsigned int flags) {
690,691c749
< void dadfs_destory_inode(struct inode *inode)
< {
---
> void dadfs_destory_inode(struct inode *inode) {
694c752,753
<   printk(KERN_INFO "Freeing private data of inode %p (%lu)\n",
---
>     printk(KERN_INFO
>     "Freeing private data of inode %p (%lu)\n",
699,700c758
< static void dadfs_put_super(struct super_block *sb)
< {
---
> static void dadfs_put_super(struct super_block *sb) {
712,713c770
< static int dadfs_load_journal(struct super_block *sb, int devnum)
< {
---
> static int dadfs_load_journal(struct super_block *sb, int devnum) {
722d778
<   printk(KERN_INFO "Journal device is: %s\n", __bdevname(dev, b));
724a781,782
>     printk(KERN_INFO
>     "Journal device is: %s\n", bdevname(bdev, b));
733c791,792
<       printk(KERN_ERR "Can't load journal\n");
---
>         printk(KERN_ERR
>         "Can't load journal\n");
742,743c801,802
< static int dadfs_sb_load_journal(struct super_block *sb, struct inode *inode)
< {
---
>
> static int dadfs_sb_load_journal(struct super_block *sb, struct inode *inode)
{
749c808,809
<       printk(KERN_ERR "Can't load journal\n");
---
>         printk(KERN_ERR
>         "Can't load journal\n");
765,766c825,826
< static int dadfs_parse_options(struct super_block *sb, char *options)
< {
---
```

```
>
> static int dadfs_parse_options(struct super_block *sb, char *options) {
782c842,843
<               printk(KERN_INFO "Loading journal devnum: %i\n", arg);
---
>                 printk(KERN_INFO
>                 "Loading journal devnum: %i\n", arg);
787,788c848
<           case DADFS_OPT_JOURNAL_PATH:
<           {
---
>             case DADFS_OPT_JOURNAL_PATH: {
796c856,857
<                   printk(KERN_ERR "could not find journal device path: error
%d\n", ret);
---
>                     printk(KERN_ERR
>                     "could not find journal device path: error %d\n", ret);
825,826c886
< int dadfs_fill_super(struct super_block *sb, void *data, int silent)
< {
---
> int dadfs_fill_super(struct super_block *sb, void *data, int silent) {
837c897,898
<   printk(KERN_INFO "The magic number obtained in disk is: [%llu]\n",
---
>     printk(KERN_INFO
>     "The magic number obtained in disk is: [%llu]\n",
913,914c974
<                   void *data)
< {
---
>                                   void *data) {
915a976,977
>     printk(KERN_INFO
>     "Hello World\n");
920c982,983
<       printk(KERN_ERR "Error mounting dadfs");
---
>         printk(KERN_ERR
>     "Error mounting dadfs");
922c985,986
<       printk(KERN_INFO "dadfs is succesfully mounted on [%s]\n",
---
>     printk(KERN_INFO
>     "dadfs is succesfully mounted on [%s]\n",
928,929c992
< static void dadfs_kill_superblock(struct super_block *sb)
< {
---
> static void dadfs_kill_superblock(struct super_block *sb) {
947,948c1010
< static int dadfs_init(void)
< {
---
> static int dadfs_init(void) {
962c1024,1025
<       printk(KERN_INFO "Sucessfully registered dadfs\n");
```

```
---
>         printk(KERN_INFO
>     "Sucessfully registered dadfs\n");
964c1027,1028
<       printk(KERN_ERR "Failed to register dadfs. Error:[%d]", ret);
---
>     printk(KERN_ERR
>     "Failed to register dadfs. Error:[%d]", ret);
969,970c1033
< static void dadfs_exit(void)
< {
---
> static void dadfs_exit(void) {
977c1040,1041
<       printk(KERN_INFO "Sucessfully unregistered dadfs\n");
---
>         printk(KERN_INFO
>     "Sucessfully unregistered dadfs\n");
979c1043,1044
<       printk(KERN_ERR "Failed to unregister dadfs. Error:[%d]",
---
>     printk(KERN_ERR
>     "Failed to unregister dadfs. Error:[%d]",
```

Reference

[1] https://docs.oracle.com/cd/E19253-01/817-5789/emjjr/index.html

[2] What Is A Kernel Module? (die.net)

[3] Compiling Kernel Modules (tldp.org)