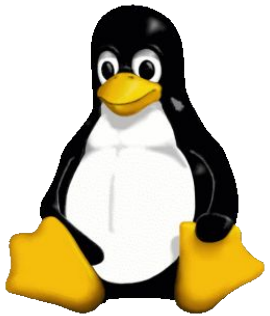


Linux Kernel Programming

Maruthi S. Inukonda

02 April 2019

License

Agenda

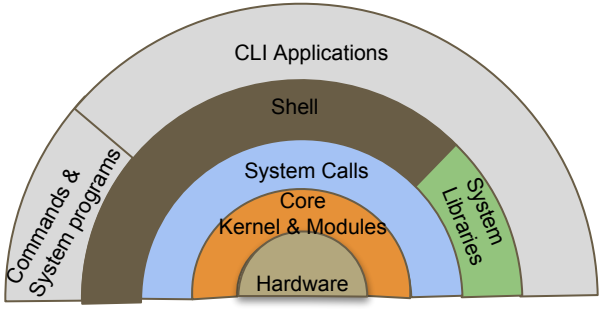
- Linux Architecture (Recap)
- Core Kernel and Modules (Recap)
- Simple Kernel Module
- Simple Character Device Driver

Linux Architecture, Core Kernel and Modules (Recap)

Excerpt from “Linux - The Beginning”, “Linux Server Administration” slides

Linux Architecture - Command Line Interface (CLI)

- Hardware
  - ➢ CPU, Memory, Disk, Graphics, Network, etc
- Core Kernel & Modules
  - ➢ Process, Memory, File, Network subsystems, Device drivers
- System Calls
  - ➢ read, write, fork, exec, clone, etc
- System Libraries
  - ➢ libc, libpthread, etc
- Commands & System programs
  - ➢ cd, ls, mkdir, top, vi, gcc, etc
- Command Line Interface (CLI) (Shell)
  - ➢ bash, sh, etc
- Command line applications
  - ➢ pine, git, gdb, etc



Core Kernel

- Boot loader is the first program loaded by firmware (BIOS or UEFI)
- Core Kernel (in `/boot/vmlinuz-<version>`) is a program loaded by boot loader (grub)
- Kernel always runs in privileged mode in kernel space.
- The Core kernel is kernel code packaged into the vmlinuz file in /boot.
- To find running kernel's version use, `uname -r`

```
$ uname -r
4.4.0-31-generic

$ ls -lR /boot/grub
-r--r--r-- 1 root root     8432 Nov 14  2016 grub.cfg
drwxr-xr-x 2 root root    12288 Nov 14  2016 i386-pc
...

$ ls -lR /boot
-rw-r--r-- 1 root root   189558 Jul 13  2016 config-4.4.0-31-generic
-rw-r--r-- 1 root root 35907255 Nov 14  2016 initrd.img-4.4.0-31-generic
-rw------- 1 root root  3866473 Jul 13  2016 System.map-4.4.0-31-generic
-rw-r--r-- 1 root root  7047520 Nov 14  2016 vmlinuz-4.4.0-31-generic
...
```

Kernel Modules

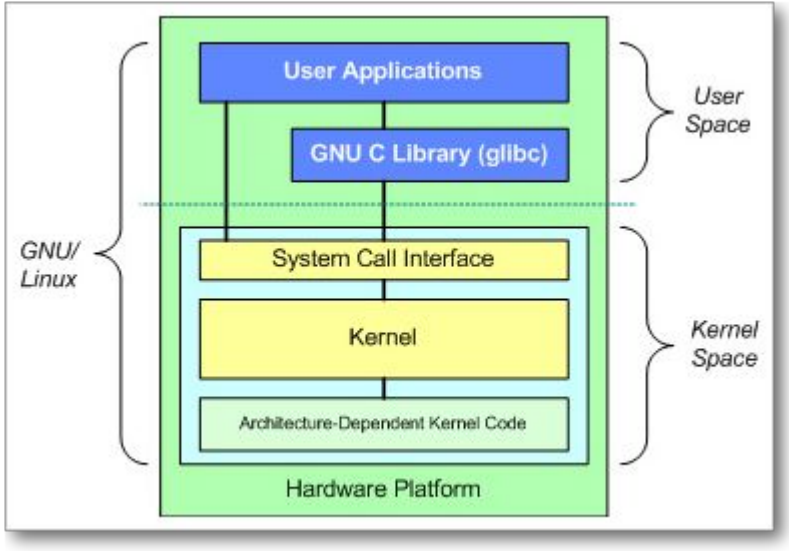
- Loadable modules having device drivers loaded by hotplug of devices.
- Kernel modules also run in privileged mode in kernel space.
- To list loaded kernel modules, use `lsmod`
- To find all modules in /lib/modules/<kernel-version>/kernel/drivers/

```
$ lsmod
Module                  Size  Used by
...
drm                   401408  6 drm_kms_helper,i915
...

$ modinfo drm
filename:       /lib/modules/4.4.0-31-generic/kernel/drivers/gpu/drm/drm.ko
license:        GPL and additional rights
description:    DRM shared core routines
author:         ...

$ ls /lib/modules/`uname -r`/kernel/drivers/gpu/drm.ko
/lib/modules/4.4.0-31-generic/kernel/drivers/gpu/drm.ko
...
```

System calls

- Entry points into the kernel.
- C language APIs.
- About 400 system calls
  - open(), read(), write(), close(), ioctl()
  - fork(), wait(), clone()
  - socket(), connect(), accept(), shutdown()
  - mmap(), munmap(), fadvise()
  - ...
- Using system calls in your program directly makes it
  - portable across Unices.
  - non-portable across Windows/Linux.

```
$ man syscalls

$ uname -o
GNU/Linux
```



Loading Kernel Modules

- To load a kernel module and its dependencies from standard path, use `modprobe -v <name>`
- To load a kernel module from any path, use `insmod <path_to_ko>`

```
# modprobe -v kvm_intel
insmod /lib/modules/4.15.0-42-generic/kernel/virt/lib/irqbypass.ko
insmod /lib/modules/4.15.0-42-generic/kernel/arch/x86/kvm/kvm.ko
insmod /lib/modules/4.15.0-42-generic/kernel/arch/x86/kvm/kvm-intel.ko

# insmod ~/AllCode/Maruthi/lkd/01.modules/01.mykmod/kernel/mykmod.ko

# lsmod | grep kvm
Module                  Size  Used by
...
kvm_intel             217088  0
kvm                   598016  1 kvm_intel
irqbypass             16384   1 kvm
...
mykmod                16384   0
...
```

Unloading Kernel Modules

- To unload a kernel module and its dependencies, use `modprobe -vr <name>`
- To unload a kernel module, use `rmmod <name>`

```
# modprobe -vr kvm_intel
rmmod kvm_intel
rmmod kvm
rmmod irqbypass

# rmmod mykmod
```

Simple Kernel Module

Module info data structure

- Module info is a structure to let the module infrastructure in core kernel know some useful information about module.
- Use predefined macros to initialize key parameters.
- Viz. `MODULE_DESCRIPTION, MODULE_AUTHOR, MODULE_LICENSE`

```
$ vi mykmod_main.c
#include <linux/module.h>
...

MODULE_DESCRIPTION("My kernel module - mykmod");
MODULE_AUTHOR("maruthisi.inukonda [at] gmail.com");
MODULE_LICENSE("GPL");

...
```

Module load/unload hooks

- Function registered using `module_init()` is called during `insmod`
- Function registered using `module_exit()` is called during rmmod

```
$ vi mykmod_main.c
#include <linux/module.h>
#include <linux/init.h>

...

static int mykmod_init_module(void);
static void mykmod_cleanup_module(void);

module_init(mykmod_init_module);
module_exit(mykmod_cleanup_module);

...
```

```
static int mykmod_init_module(void)
{
     printk(KERN_INFO "mykmod loaded\n");
     // Do any initializations here

     return 0;
}

static void mykmod_cleanup_module(void)
{
     // Do any de-initializations here

     printk(KERN_WARNING "mykmod unloaded\n");

     return;
}
...
```

Kbuild Makefile

- "kbuild" is the build system used by the Linux kernel.
- Modules must use kbuild to stay compatible with changes in the build infrastructure and to pick up the right flags to "gcc."

```
$ vi Makefile
# If KERNELRELEASE is defined, we've been invoked from the
# kernel build system and can use its language
ifneq ($(KERNELRELEASE),)
     obj-m  := mykmod.o
     mykmod-objs := mykmod_main.o
else
     KERNELRELEASE  ?= /lib/modules/$(shell uname -r)/build
     PWD  := $(shell pwd)
default:
     $(MAKE) -C $(KERNELRELEASE) M=$(PWD) modules
endif

clean:
     @rm -rf *.o *.ko Module.* modules.order .*.cmd *.mod.c \
     *.mod.c .tmp_versions .cache.mk
```

Build, Load and Unload

- Build the module using make.
- Load the module using insmod <module.ko>.
- Check the kernel logs using `dmesg`.
- Unload the module using rmmod <module>
- To clean the build use `make clean`.

```
$ make

# insmod mykmod.ko

$ dmesg
...
[612297.907010] mykmod loaded

$ lsmod
Module                  Size  Used by
...
mykmod                 16384  0
…

# rmmod mykmod

$ dmesg
...
[612346.334620] mykmod unloaded

$ make clean
```

Drivers, Devices (Recap)

Excerpt from “Linux Server Admin” slides

Drivers

- Driver is a kernel module that manages devices (aka Device driver).
- Two types : “Character device drivers” or “Block device drivers”.
- Devices could be real or pseudo.
- Each driver is uniquely identified using major no.
- List all drivers using `cat /proc/devices`

```
$ cat /proc/devices
Character devices:
  4 tty
...
 81 video4linux
...
136 pts
...
Block devices:
...
  7 loop
  8 sd
...
253 device-mapper
```

Character devices

- Driver creates a character device special file for each character device instance (or we create manually using mknod)
  - Eg. keyboard, mouse, many pseudo devices.
- Accessible in unit of 1B
- List all character devices using `ls -l /dev/ | grep ^c`
- Each device instance is uniquely identified using major,minor number pair.

```
$ ls -l /dev/ | grep ^c
crw-rw-rw-  1 root root       1,   3 Dec  7 05:53 null
...
crw--w----  1  maruthisi tty  136,   0 Dec  8 06:56 pts/0
...
crw--w----  1 root tty        4,   0 Dec  7 05:53 tty0
...
crw-rw----+ 1 root video      81,   0 Dec  7 05:53 video0
crw-rw-rw-  1 root root       1,   5 Dec  7 05:53 zero
```

Block devices

- Driver creates a block device special file for each block device instance (or we create manually using `mknod`)
  - Eg. Disk, Tape, CD/DVD, many pseudo devices.
- Accessible in units of 512B, 1KiB, 4KiB.
- List all character devices using `ls -l /dev/ | grep ^b` or `lsblk -pa`
- Each device instance is uniquely identified using major,minor number pair.

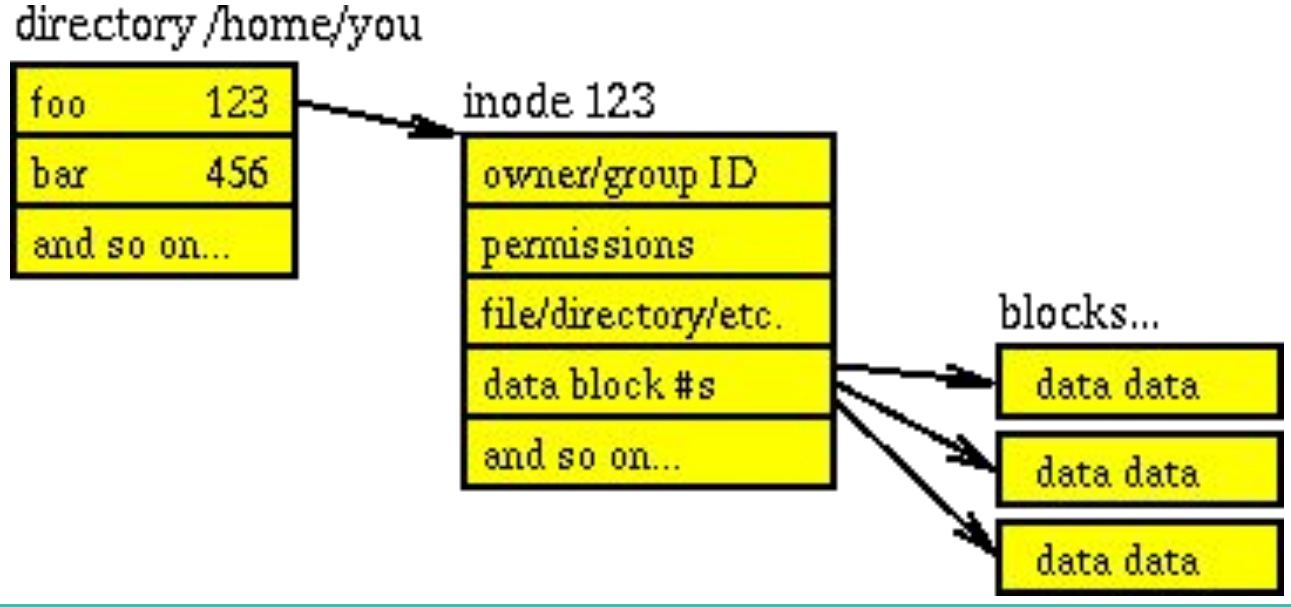
```
$ ls -l /dev/ | grep ^b
brw-rw----  1 root disk      7,   0 Apr  2 03:09 loop0
...
brw-rw----  1 root disk      8,   0 Apr  2 03:09 sda
brw-rw----  1 root disk      8,   1 Apr  2 03:09 sda1
...
```

File, Dirent, I-node (Recap)

Excerpt from “Linux Commands” slides

On-disk Index node and Directory Entry

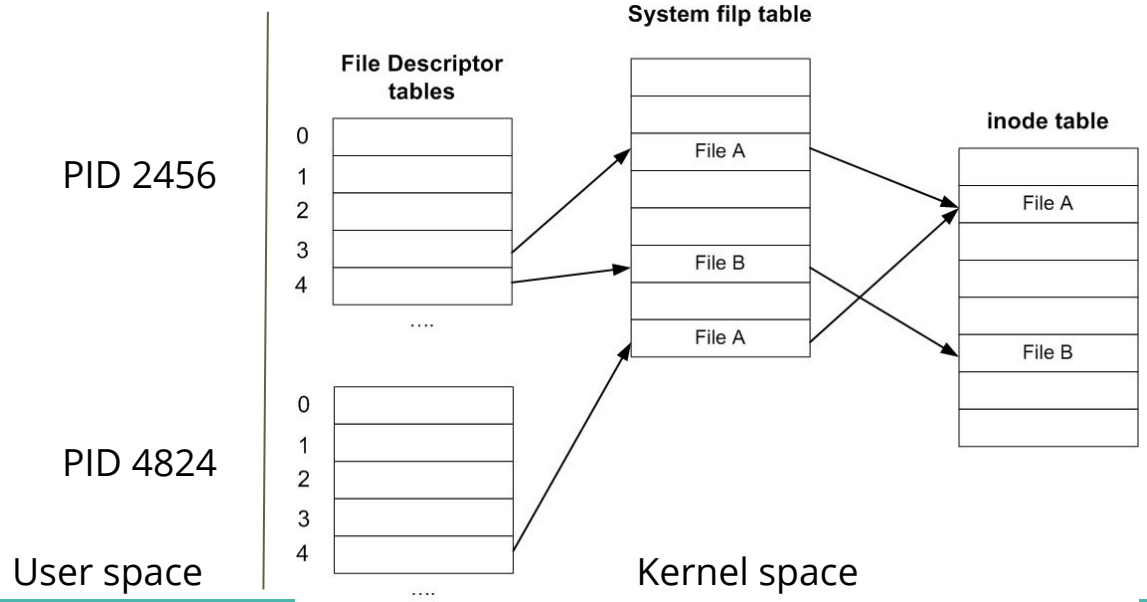
- Every file has few ondisk data structures for metadata.
- Index node (inode) with a unique number (ino). Filename is not part of inode.
- Directory entries (dirent) which stores the file name, and inode number.



Kernel Data Structures

file, inode data structures

- `file` is an in-core structure to represent a file opened by a process. One such data structure exists per process, per file descriptor (1 per `open(2)`).
- `inode` is an in-core structure to represent a ondisk inode opened by any process in the system. It is a shared data structure across processes.



Virtual File System (VFS) components

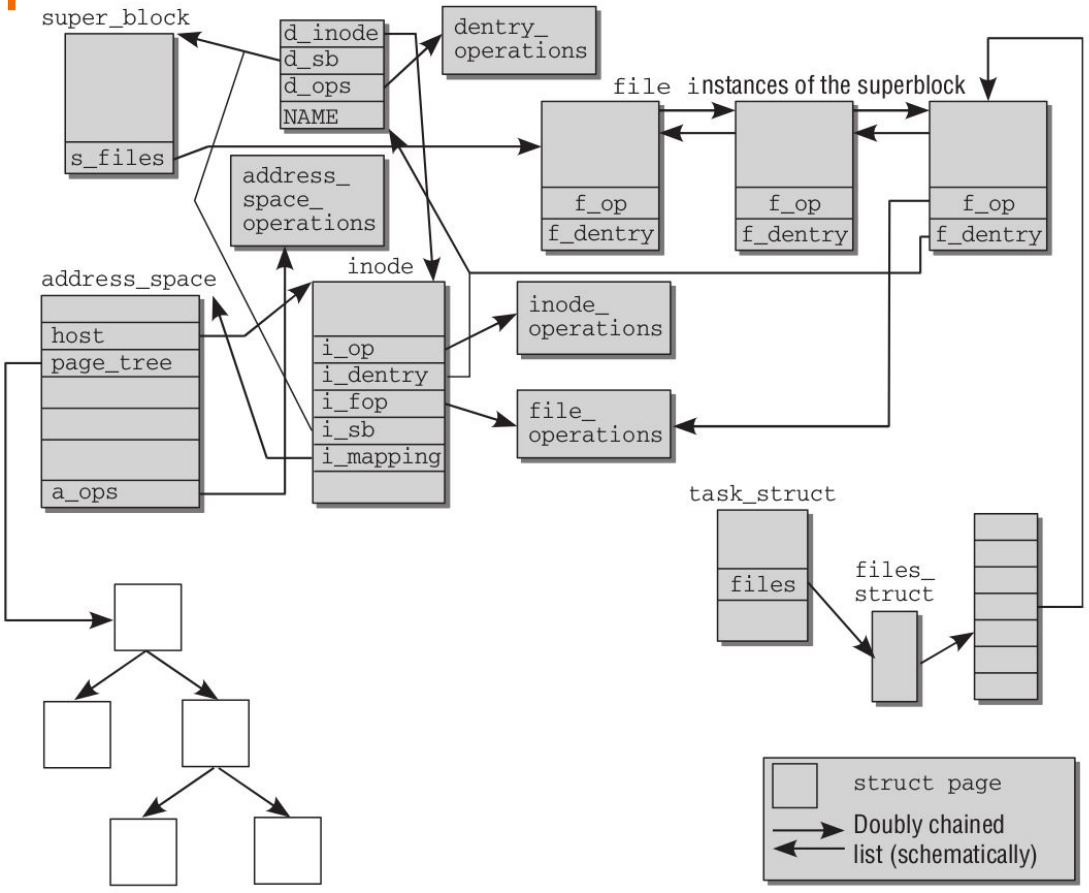


Image courtesy: Professional Linux Kernel Architecture, Wolfgang Mauerer

Simple Pseudo Character Device Driver

File Operations

- An instance of `file_operations` is used to enlist all supported operations on the driver's device special file.

```
$ vi mykmod_main.c
...
#include <linux/fs.h>
#include <linux/cdev.h>
...

static int mykmod_open(struct inode *inode, struct file *filp);
static int mykmod_close(struct inode *inode, struct file *filp);

static struct file_operations mykmod_fops = {
    .owner   = THIS_MODULE,          /* owner (struct module *) */
    .open    = mykmod_open,          /* open */
    .release  = mykmod_close,        /* release */
};
```

Registering and Unregistering character device driver

- The driver name and file_operations structure should passed to register.
- The major number and name should be passed unregister.

```
#define MYKMOD_DEV_MAJOR 0   // Dynamically allocate major no
int mykmod_dev_major;

static int mykmod_init_module(void) {
   ...
   mykmod_dev_major =
register_chrdev(MYKMOD_DEV_MAJOR,"mykmod",&mykmod_fops);
   ...
}

static int mykmod_init_module(void) {
   ...
   unregister_chrdev(mykmod_dev_major,"mykmod");
   ...
}
```

File Operations - open, release

- `.open and .release` function pointers need to point to respective call backs in the driver.
- They get invoked whenever `open(2)` are `close(2)` are called from userspace respectively.

```
static int
mykmod_open(struct inode *inodep, struct file *filep)
{
     printk("mykmod_open: inodep=%p filep=%p\n", inodep, filep);
     return 0;
}

static int
mykmod_close(struct inode *inodep, struct file *filep)
{
     printk("mykmod_close: inodep=%p filep=%p\n", inodep, filep);
     return 0;
}
```

Userspace utility

- In a c program, make `open()` and `close()` system calls, to get calls into the driver's operations.

```
$ vi mykmod_test.c
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>

int main(int argc, char *argv[])
{
      int fd;
      fd = open(argv[1], O_RDWR);
      if (fd<=0) { ... }

      close(fd);

      return 0;
}

$ gcc -o mykmod_test mykmod_test.c
```

Build, Load, Test and Unload

- Build, Load the module.
- Note the major number from `/proc/devices` or from `dmesg`
- Create a device special file with driver's major number and any minor number (0-255) using `mknod <device_special_file> c <majorno> <minorno>`

```
$ make
# insmod mykmod.ko
...
[613212.687562] mykmod loaded
[613212.687570] register character device 510

# mknod /tmp/mydsf c 510 0
$ ls -l /tmp/mydsf
crw-r--r-- 1 root root 510, 0 Apr  2 12:11 /tmp/mydsf

# ./mykmod_test /tmp/mydsf
$ dmesg
...
[613552.446487] mykmod_open: inodep=00000000aa3212c8 filep=00000000a9fc9a4a
[613552.446500] mykmod_close: inodep=00000000aa3212c8 filep=00000000a9fc9a4a
```

References

References

- Understanding the Linux Kernel, 3rd Edition, Bovet & Cesati, Oreilly.
- Linux Device Drivers, 3rd Edition, Corbet, Rubini & Hartman, Oreilly.
- Linux Kernel Development, Robert Love, Pearson Education.

Q & A