Course Booklet for Linux Device Drivers

The Practical Approach to Linux Drivers

By Emertxe

Version 2.5 (December 18, 2014)

Course Email: embedded.courses@emertxe.com

Contents

Preface

Linux Device Driver course module covers the initators and some detail insights of writing a Device Driver in Linux, focussed and aligned towards the industry perspective. By the end of the module, students are expected to have made their concepts very clear and become comfortable interpreting datasheets, and writing any character device driver from software perspective

The theory duration for this module roughly spans 2 weeks, with practicals spanning another 2 weeks, apart from the parallel lab sessions, and assignments

Pre-requisites for this module are: BE/BTech (4th semester onwards), Advanced C Programming, Data Structures, Linux Internals

This booklet serves as a workbook guide, as we go along through the Device Driver course module at Emertxe Information Technologies Pvt Ltd. It will provide the enough information pointers to remember, alongwith the references to detailed relevant materials. It will provide a ground for practice and for writing your personalized notes

As part of this, it would cover: Mastering the basics, Getting onto your first device driver, Writing your own real-life device driver, various debugging techniques, and many other industry relevant topics. As an overall experience, it will take you through all the nitty-gritties of device driver know-hows & writing through a well organized set of examples and exercises

The appendix will contain references and details on the topics not directly related to the module but nevertheless relevant to it

Finally, there are two appendices dedicated to a set of assignments & projects,respectively, well sorted and graded, which gets reviewed after every batch

Special about this module (and this booklet):

- Concentrates on the in-depth concepts, making you more confident about you and your skills
- Well packed and graded set of assignments and project for practice
- Focus on the way industry uses device drivers, making you more confident at interviews and getting a job
- Regular mutual feedbacks to improvise and tune the module as per the audience
- Regular updates from the industry to keep abreast with induustry
- Learn many more things, as added benefits than just the module, like:
  - Get hands-on to work on Linux
  - Become expert on various tools
  - Increase your productivity

Chapter 1

Introduction

1.1 Familiarity Check

Notes:

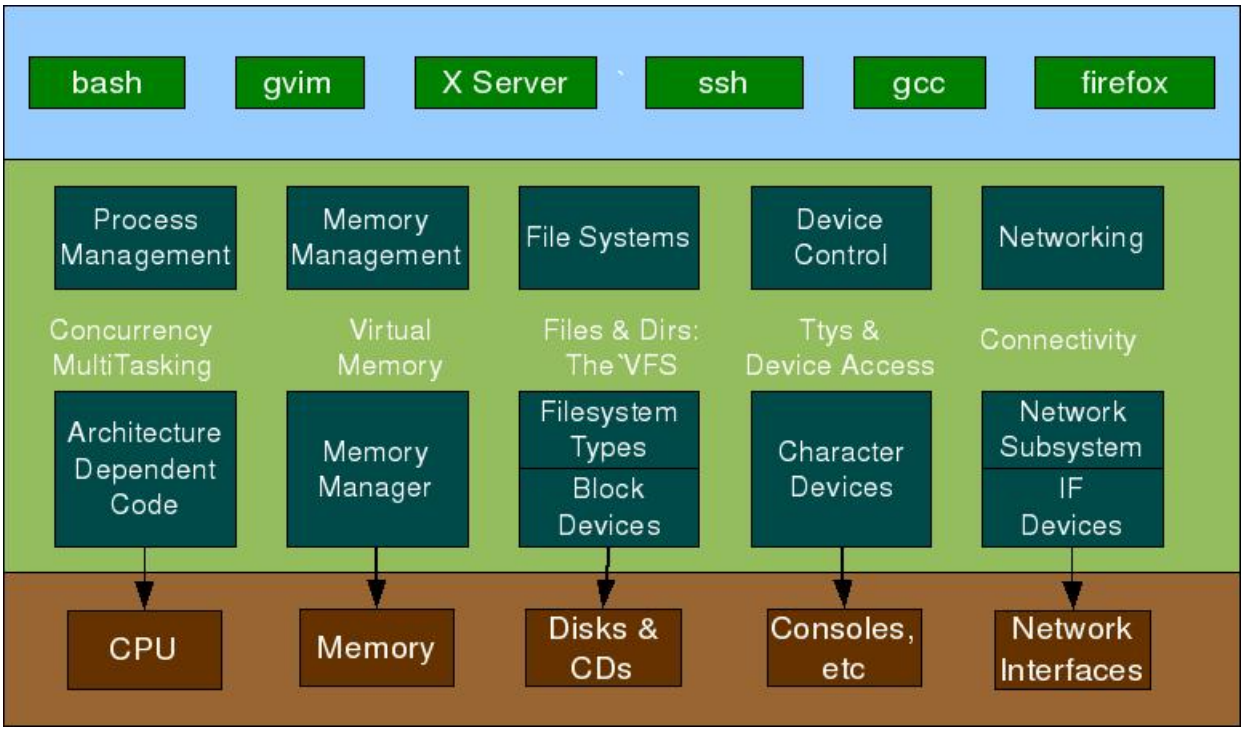
- Good C & Programming Skills
- Linux & the Filesytem
- Files: Regular, Special, Device
- Toolchain
- Make & Makefiles
- Kernel Sources (Location & Building)

1.1.1 Glossary

- **API:** Set of functions, procedures, methods, classes or protocols that an operating system, library or service provides to support requests made by computer programs
- **Card:** Also called expansion or adaptor card. Printed Circuit Board that can be inserted into a slot of a computer motherboard to add additional functionality to a computer system
- **Device:** Hardware to achieve a specified functionality of a computer system
- **Driver:** Computer program which controls, manages, operates a device or an another program. The former is called a device driver
- **File:** Block of arbitrary information. Chunks of organized information. And hence, in Linux, everything a user interacts with at the system level, is a file
- **Filesystem:** The way a computer organizes, names, stores and manipulates files is globally referred to as its file system
- **Interface:** An abstraction that an entity provides of itself to the outside, separating the methods of external communication from internal operation. This allows it to be internally modified without affecting the way outside entities interact with it, as well as provide multiple abstractions of itself
- **Kernel:** Most commonly used name for the Linux operating system
- **Module:** A Linux driver which is hot plug-n-play, or in other words which can be dynamically loaded into and unloaded from the kernel
- **Port:** I/O Device. Contrast memory-mapped to port-mapped
- **Protocol:** Convention or Standard that controls or enables the connection, communication, and data transfer between two computing endpoints. It could be implemented in hardware, software or both
- **Slot:** Also, called expansion slot. Connector on computer motherboard for inserting card to add additional functionality to a computer system
- **System Call:** Mechanism used by an application program to request service from the kernel

Notes:

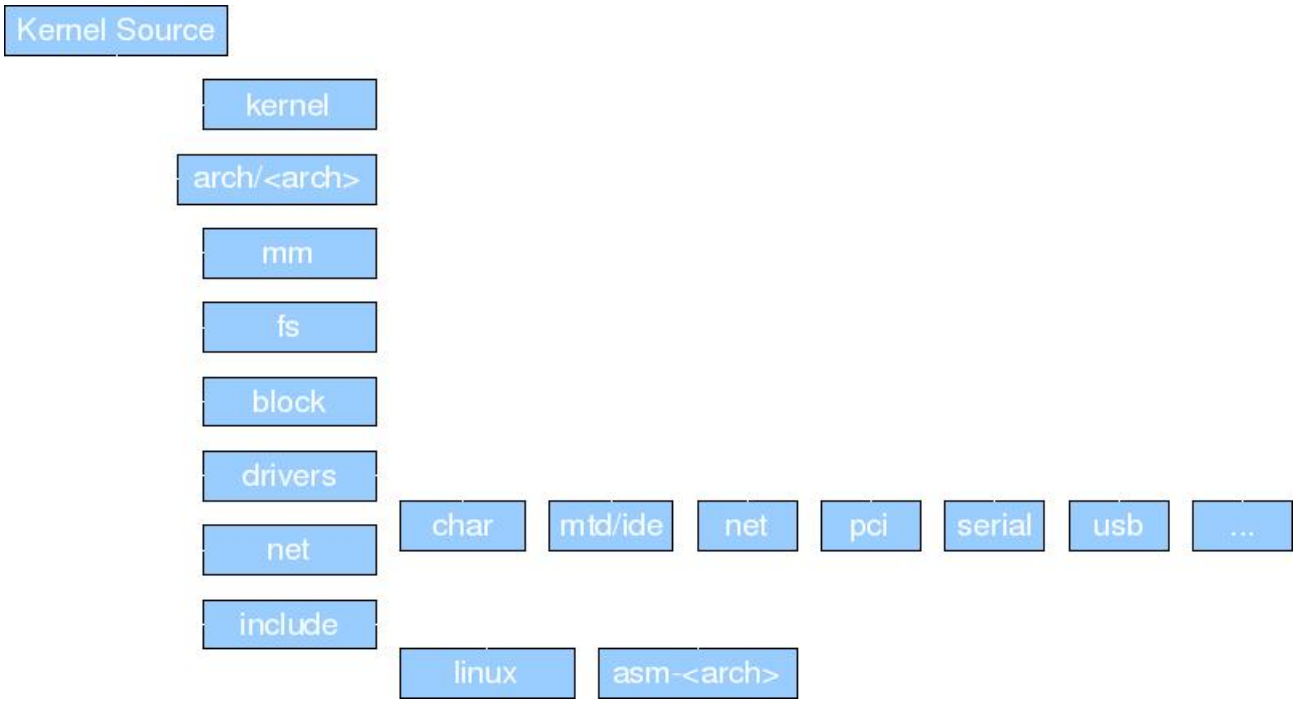
1.2 Ecosystem



- CPU subsystem
- Memory subsystem
- File subsystem and Block drivers
- Character drivers
- Network subsystem & Network drivers

Notes:

1.3 The Kernel



- Source Path: /usr/src/linux
- Headers Path:/usr/src/linux/include
- Makefile:/usr/src/linux/Makefile

Notes:

- Modules Path:/lib/modules/<kernel version>/kernel
- Module Configuration:/etc/modprobe.conf
- Kernel Windows:/proc & /sys
- System Logs:/var/log/messages

Notes:

1.4 The Basic Commands

- lsmod
- insmod <module file>
- modprobe <module name>
- rmmod <module name>
- dmesg
- objdump <-d|-D> <object or exe file>
- nm <object or exe file>
- cat /proc/...

Notes:

1.5 The First Driver

1.5.1 Module's constructor and destructor

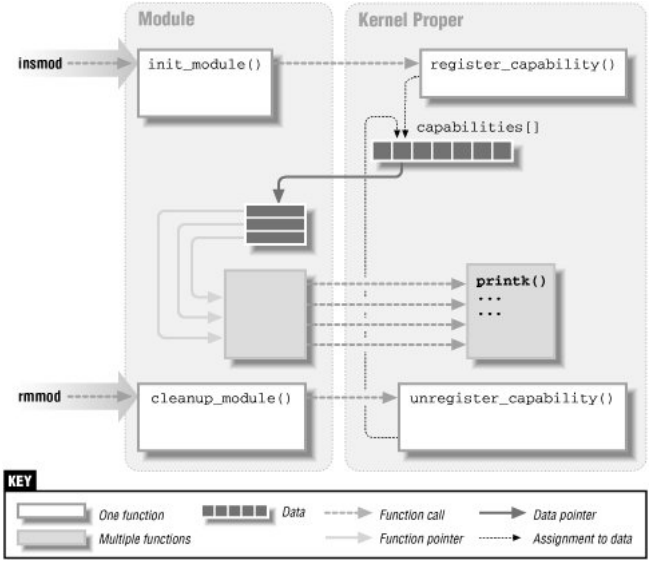
- init_module
- cleanup_module

Notes:

1.5.2 Kernel's printf

Notes:

```
KERN_EMERG      "<0>"   /* system is unusable */
KERN_ALERT      "<1>"   /* action must be taken immediately */
KERN_CRIT       "<2>"   /* critical conditions */
KERN_ERR        "<3>"   /* error conditions */
KERN_WARNING    "<4>"   /* warning conditions */
KERN_NOTICE     "<5>"   /* normal but significant condition */
KERN_INFO       "<6>"   /* informational */
KERN_DEBUG      "<7>"   /* debug-level messages */
```



1.5.3 The Kernel Headers

Notes:

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

1.5.4 The Other Ornaments

Notes:

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("<your name>");
MODULE_DESCRIPTION("First Device Driver");
```

1.5.5 Building the Module

- Building under the Source Tree
- Invoking the Kernel's Build System

The Makefile

```
ifneq (${KERNELRELEASE},)
    obj-m += <module>.o
else
    KERNEL_SOURCE := <kernel source directory path>
    PWD := $(shell pwd)
default:
    $(MAKE) -C ${KERNEL_SOURCE} SUBDIRS=$(PWD) modules
clean:
    $(MAKE) -C ${KERNEL_SOURCE} SUBDIRS=$(PWD) clean
endif
```

Chapter 2

Character Drivers

2.1 Major & Minor numbers

Notes:

```
<linux/types.h>
dev_t: 12 & 20
<linux/kdev_t.h>
MAJOR(dev_t dev);
MINOR(dev_t dev);
MKDEV(int major, int minor);
```

2.1.1 Registering & Unregistering a device

Notes:

```
<linux/fs.h>
int register_chrdev_region(dev_t first, unsigned int count, char *name);
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,
    unsigned int cnt, char *name);
void unregister_chrdev_region(dev_t first, unsigned int count);
```

2.2 File operations

```
<linux/fs.h>
```

Notes:

2.2.1 The struct file_operations

```
struct module owner = THIS_MODULE; / linux/module.h> */
int (*open)(struct inode *, struct file *);
int (*release)(struct inode *, struct file *);
ssize_t (*read)(struct file *, char __user *, size_t, loff_t *);
ssize_t (*write)(struct file *, const char __user *, size_t, loff_t *);
loff_t (*llseek)(struct file *, loff_t, int);
int (*ioctl)(struct inode *, struct file *, unsigned int, unsigned long);
```

Notes:

2.2.2 The struct file

```
mode_t f_mode
loff_t f_pos
unsigned int f_flags
struct file_operations *f_op
void *private_data
```

Notes:

2.2.3 The struct inode

```
unsigned int iminor(struct inode *);
unsigned int imajor(struct inode *);
```

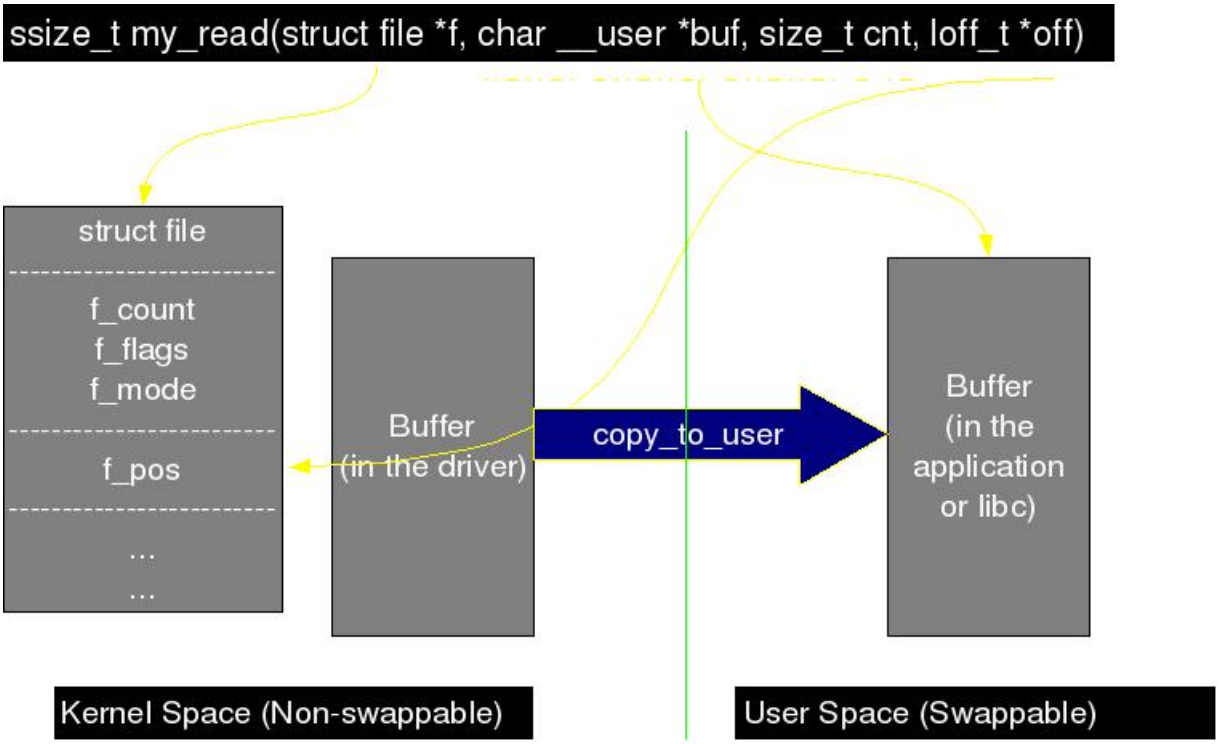
Notes:

2.2.4 The fops Registration

```
struct cdev my_cdev;
cdev_init(&my_cdev, &my_fops);
my_cdev.owner = THIS_MODULE;
int cdev_add(struct cdev *cdev, dev_t num, unsigned int count);
void cdev_del(struct cdev *cdev);
```

Notes:

2.2.5 The read & write flows



Notes:

Null & Memory Character Driver

Notes:

2.2.6 The ioctl

Notes:

```
<linux/ioctl.h>
#define CMD1 _IO('f', 1)
#define CMD2 _IOR('f', 2, IoctlArgType)
#define CMD3 _IOW('f', 3, IoctlArgType)
#define CMD4 _IOWR('f', 4, IoctlArgType)
```

Notes:

2.3 Dynamic Device Allocation & udev

Notes:

```
struct class *class_create(struct module *owner, char *name);
void class_destroy(struct class *cl);
struct device *device_create(struct class *cl, NULL,
    dev_t devnum, NULL, const char *fmt, ...);
void device_destroy(struct class *cl, dev_t devnum);
```

Notes:

2.4 Module Parameters

Notes:

```
<linux/moduleparam.h>
module_param(name, type, perm)
module_param_array(name, type, num, perm)
```

Notes:

2.5 Exercises:

Chapter 3

Memory and Hardware

3.1 Virtual Address Management

```
#include <linux/gfp.h>
unsigned long __get_free_pages(gfp_t gfp_mask, unsigned int order);
void free_pages(unsigned long addr, unsigned int order);

#include <linux/slab.h>
void *kmalloc(size_t size, gfp_t flags);
void kfree(const void *addr);

#include <linux/vmalloc.h>
void *vmalloc(unsigned long size);
void vfree(void *addr);
```

Notes:

3.2 Virtual Address for Bus/IO Address

```
#include <asm/io.h>
void *ioremap(unsigned long offset, unsigned long size);
void iounmap(void *addr);
```

Notes:

3.3 I/O Memory Access

```
<asm/io.h>
unsigned int ioread[8|16|32](void *addr);
unsigned int iowrite[8|16|32](u[8|16|32] value, void *addr);
```

Notes:

3.4 Barriers

```
<linux/kernel.h>
void barrier(void);
<asm/system.h>
void [r|w|]mb(void);
```

Notes:

3.5 I/O Port Access

```
<asm/io.h>
unsigned in[b|w|l](unsigned port);
void out[b|w|l](unsigned [char|short|int] value, unsigned port);
```

Notes:

3.6 The Hardware: LED circuit

3.6.1 The circuit diagram

Notes:

3.6.2 LED

Notes:

3.6.3 Shift Register

Notes:

3.6.4 MAX232

Notes:

3.6.5 UART

Notes:

3.6.6 Registers & Base Address

Notes:

3.7 Character Driver for the LED circuit

Notes:

3.8 Exercises:

Chapter 4

Debugging, Concurrency, and Time

4.1 The Debugging Options

4.1.1 Printing

- printk
- syslogd

Notes:

4.1.2 Querying

- /proc
- ioctl
- sysfs

Notes:

4.1.3 Creating proc entry's

```
/proc: <linux/proc_fs.h>
    Type: struct proc_dir_entry
    Functions:
create_proc_entry(const char *name, mode_t mode, struct proc_dir_entry *parent
  read_proc_t *read_proc, void*);
remove_proc_entry(const char *name, struct proc_dir_entry *parent);
typedef int (read_proc_t)(char *page, char **start, off_t off,
              int count, int *eof, void *data);
```

Notes:

4.1.4 Watching

- strace
- Oops

Notes:

4.2 Advanced Debugging Options

4.2.1 Debuggers

- gdb <kernel src>/vmlinux /proc/kcore
- kgdb & Remote Debugging
- kdb (oss.sgi.com)

Notes:

4.2.2 Use Mode Linux (UML)

Notes:

4.2.3 Linux Trace Toolkit

Notes: http://www.opersys.com/LTT

4.3 Concurrency

4.3.1 With Locking

Notes:

```
Semaphore: <asm/semaphore.h>
    Type: struct semaphore
    Functions: DECLARE_*, init_MUTEX, down[_trylock], up
Spin Locks: <linux/spinlock.h>
    Type: spinlock_t
    Functions: spin_lock_init, spin_[try]lock, spin_unlock
```

4.3.2 Without Locking

Notes:

```
Atomic Variables: <asm/atomic.h>
    Type: atomic_t
    Functions: ATOMIC_INIT, atomic_[set|read|add|sub|inc|dec]
Atomic Bit Operations: <asm/bitops.h>
    Functions: [set|clear|change|test*]_bit(nr, void *addr)
```

4.4 Time Keeping

4.4.1 Relative Time Keeping

Notes:

```
<linux/param.h> - HZ
<linux/jiffies.h> - jiffies & jiffies_64
    time_after, time_before, ...
    get_jiffies_64, ...
    timespec/timeval vs jiffies
Platform specific Time Stamp Counter
    <asm/msr.h> - rdtsc
    <linux/timex.h> - get_cycles
```

4.4.2 Absolute Time Keeping

Notes:

```
<linux/time.h>
    mktime(y, m, d, h, m, s)  Seconds since Epoch
    void do_gettimeofday(struct timeval *tv);
    struct timespec current_kernel_time(void);
```

4.5 Delays

4.5.1 Busy Waiting & Yielding

Notes:

```
Busy wait: cpu_relax
    while (time_before(jiffies, j1))
        cpu_relax();
Yielding: schedule/schedule_timeout
    while (time_before(jiffies, j1))
        schedule();
<linux/wait.h>: Wait Qs
    wait_queue_head_t wait;

    init_waitqueue_head(&wait);
    wait_event_interruptible_timeout(wait, 0, delay)
```

4.5.2 More Precise Delays & Sleep

Notes:

```
<linux/delay.h>
Architecture specific impl. in <asm/delay.h>
    void ndelay(unsigned long ndelays);
    void udelay(unsigned long udelays);
    void mdelay(unsigned long mdelays);
    void msleep(unsigned int millisecs);
    unsigned long msleep_interruptible(unsigned int millisecs);
    void ssleep(unsigned int secs);
```

4.6 Timers

4.6.1 Kernel Timers

Notes:

```
<linux/timer.h>
void init_timer(struct timer_list *);
struct timer_list TIMER_INITIALIZER(f, t, p);
void add_timer(struct timer_list *);
void del_timer(struct timer_list *);
int mod_timer(struct timer_list *, unsigned long);
int del_timer_sync(struct timer_list *);
```

4.6.2 Tasklets

Notes:

```
<linux/interrupt.h>
void tasklet_init(struct tasklet_struct *t, void (*func)(unsigned long), unsigned l
DECLARE_TASKLET(name, func, data);
tasklet_disable/enable(t);
tasklet_[hi_|]_schedule(t);
tasklet_kill(t);
```

4.6.3 Work Queues

Notes:

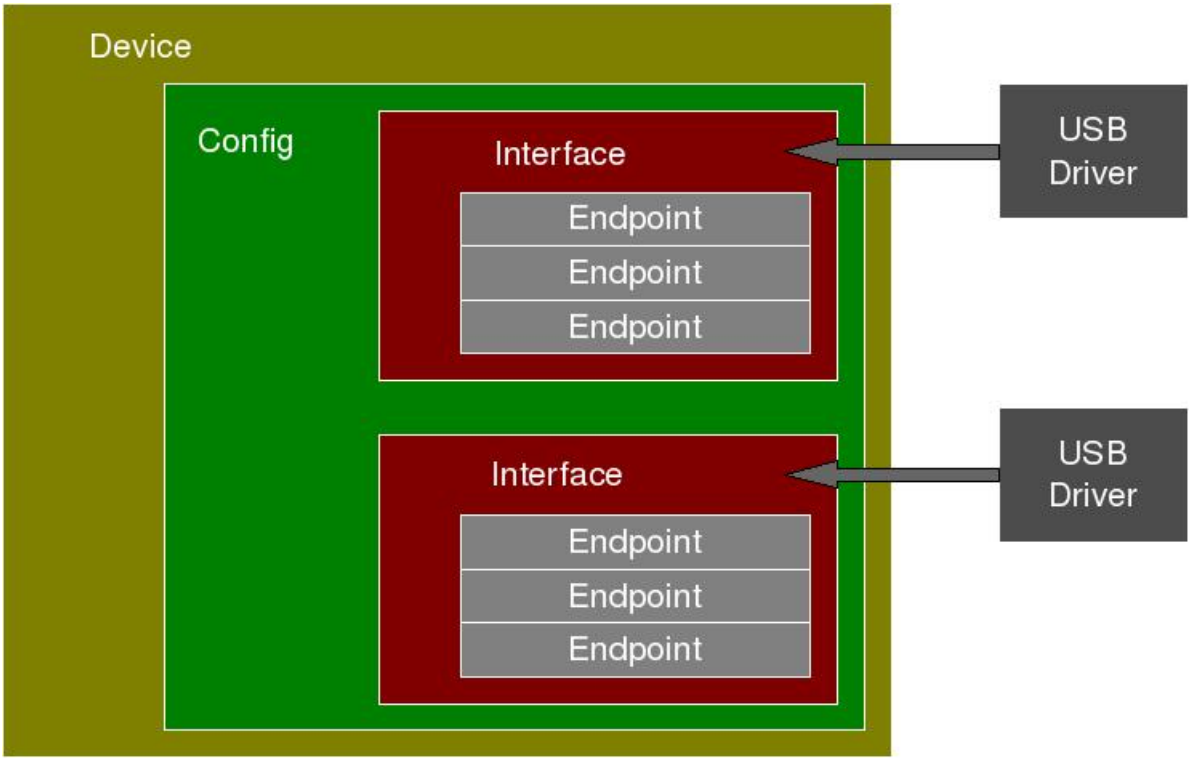
```
<linux/workqueue.h>
struct workqueue_struct *q = create_workqueue(name);
struct workqueue_struct *q = create_singlethread_workqueue(name);
DECLARE_WORK(w, void (*function)(void *), void *data);
int queue_work(q, &w);
int queue_delayed_work(q, &w, d); / int cancel_delayed_work(&w);
flush/destroy_workqueue(q);
```

Chapter 5

USB Drivers

5.1 USB Device Basics

5.1.1 USB Device Overview



Notes:

5.1.2 USB Endpoints

Notes:

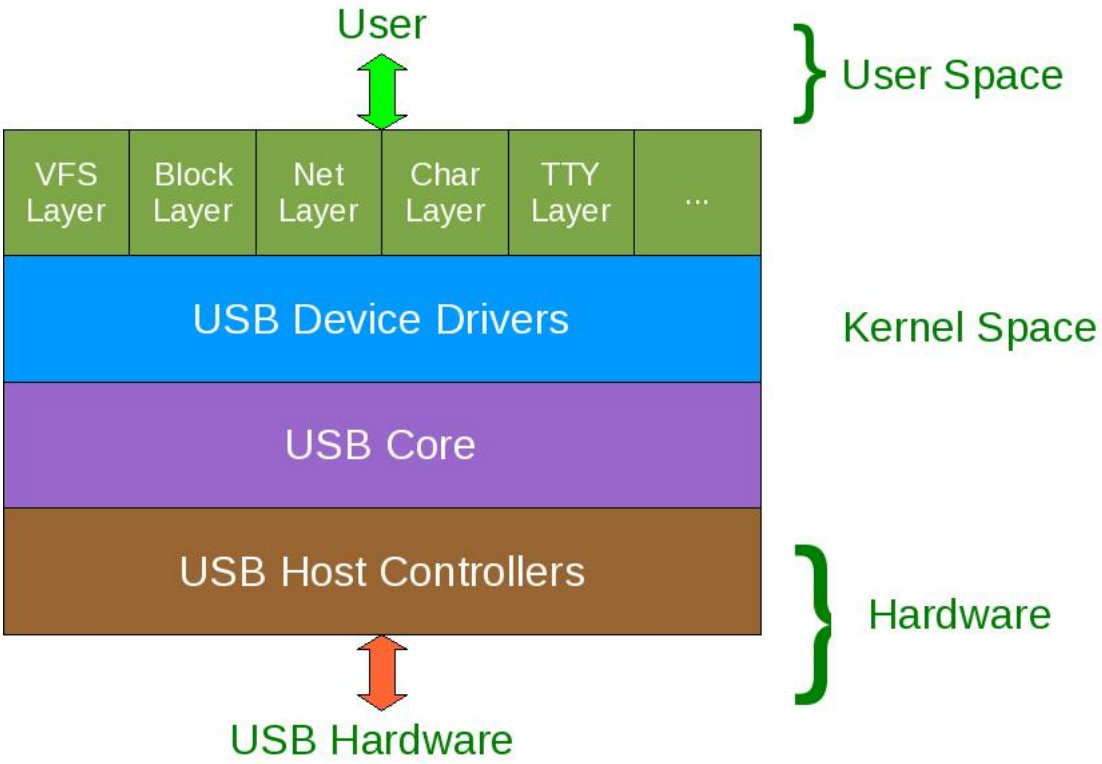
- Control
- Interrupt
- Bulk
- Isochronous

5.1.3 USB Data Structures

Notes:

```
struct usb_host_endpoint
    struct usb_endpoint_descriptor
struct usb_interface
struct usb_host_config
struct usb_device
    interface_to_usbdev
```

5.2 USB Driver Overview



Notes:

5.3 USB Core & Sysfs

Notes:

5.4 USB Driver Registration

Notes:

```
struct usb_driver
    struct module *owner;
    const char *name;
    const struct usb_device_id *id_table;
    int (*probe)(struct usb_interface *, struct usb_device_id *);
    int (*disconnect)(struct usb_interface *);
int usb_register(struct usb_driver *);
int usb_deregister(struct usb_driver *);
```

5.5 USB Device Hot-plug-ability

Notes:

```
Callback probe calls
    int usb_register_dev(intf, class);
Callback disconnect calls
    int usb_deregister_dev(intf, class);
void usb_set_intfdata(intf, void *data);
void *usb_get_intfdata(intf);
```

5.6 USB Transfers

5.6.1 struct urb: The USB Request Block (URB)

Notes:

```
struct usb_device *dev;
unsigned int pipe;
unsigned int transfer_flags;
void *transfer_buffer;
int transfer_buffer_length;
usb_complete_t complete;
int actual_length;
int status;
Pipe type specific fields
```

5.6.2 The URB Operations

Notes:

```
usb_alloc_urb / usb_free_urb
usb_fill_[int|bulk|control]_urb
usb_submit_urb
usb_unlink_urb / usb_kill_urb
```

5.6.3 The Transfer Wrappers

Notes:

```
usb_bulk_msg(dev, pipe, data, len, &act_len, timeout);
usb_control_msg(dev, pipe, req, req_type, value, index, data, size, timeout);
```

5.7 USB to Serial Device Driver

Notes:

5.8 Exercises:

Chapter 6

Interrupts

6.1 Interrupt Requests (IRQs)

Notes:

```
x86 Specific: 0x00 to 0x1F
Board Specific: 0x20 to 0xFF
<asm/interrupt.h> -> <asm/irq.h> -> irq_vectors.h
```

6.2 The APIs

Notes:

```
<linux/interrupt.h>
typedef irqreturn_t (*irq_handler_t)(int, void *);
int request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,
    const char *name, void *dev_id);
void free_irq(unsigned int irq, void *dev_id);
int can_request_irq(irq, flags);
    flags: IRQF_DISABLED, IRQF_SAMPLE_RANDOM, IRQF_SHARED, IRQF_TIMER ...
```

6.3 Miscellaneous Info

6.3.1 IRQ Control

Notes:

```
enable_irq(irq);
disable_irq(irq);
```

6.3.2 Autoprobing IRQs

Notes:

```
irqs = probe_irq_on();
/* Send an interrupt */
irq = probe_irq_off(irqs);
```

6.3.3 IRQ Handler

Notes:

```
Returning IRQ_NONE:
Returning IRQ_HANDLED:
```

6.4 Soft IRQs

Notes:

- Timer
- Network
- Block
- Tasklet
- Scheduler
- High Resolution Timer

6.5 The Two Halves

6.5.1 Top Half

Notes: Registered using request_irq

6.5.2 Bottom Half

Notes: Registered using tasklets and/or workqueues

Chapter 7

Block Drivers & File Systems

7.1 Registering & Unregistering the Block Driver

Notes:

```
<linux/fs.h>
int register_blkdev(major, name);
int unregister_blkdev(major, name);
```

7.2 Registering & Unregistering the Disk Drive

Notes:

```
<linux/genhd.h>
struct gendisk *gd = alloc_disk(minors);
del_gendisk(gd);
add_disk(gd);
struct gendisk
    int major
    int first_minor
    int minors
    char disk_name[32]
    struct block_device_operations *fops
    struct request_queue *queue
    int flags (GENHD_FL_REMOVABLE, ...)
    sector_t capacity
    void *private_data
```

7.3 The Block Device Operations

Notes:

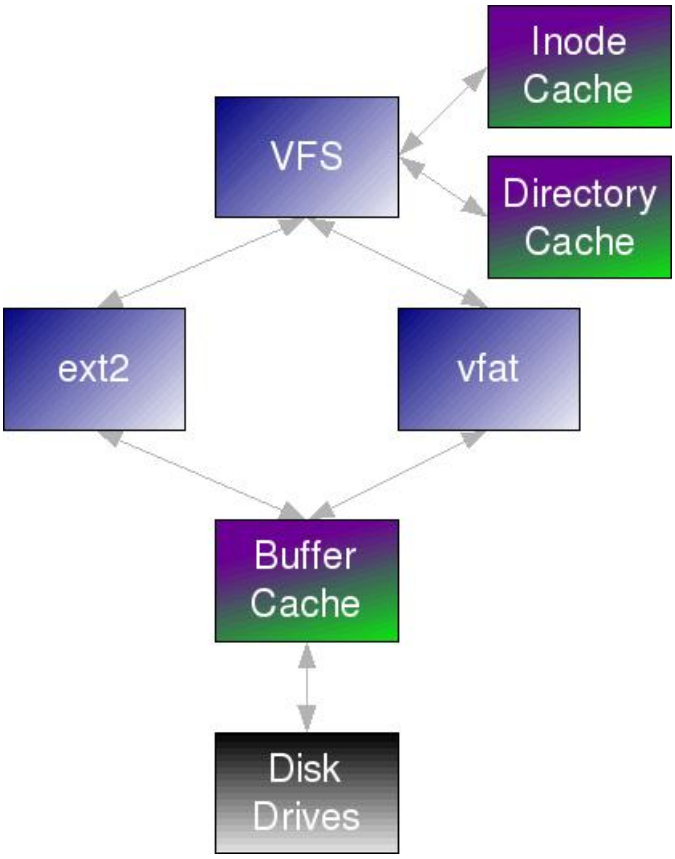
```
int open(struct inode *i, struct file *f);
int close(struct inode *i, struct file *f);
int ioctl(struct inode *i, struct file *f, cmd, arg);
int media_changed(struct gendisk *gd);
int revalidate_disk(struct gendisk *gd);
struct module *owner;
```

7.3.1 Request Queues & Processing

Notes:

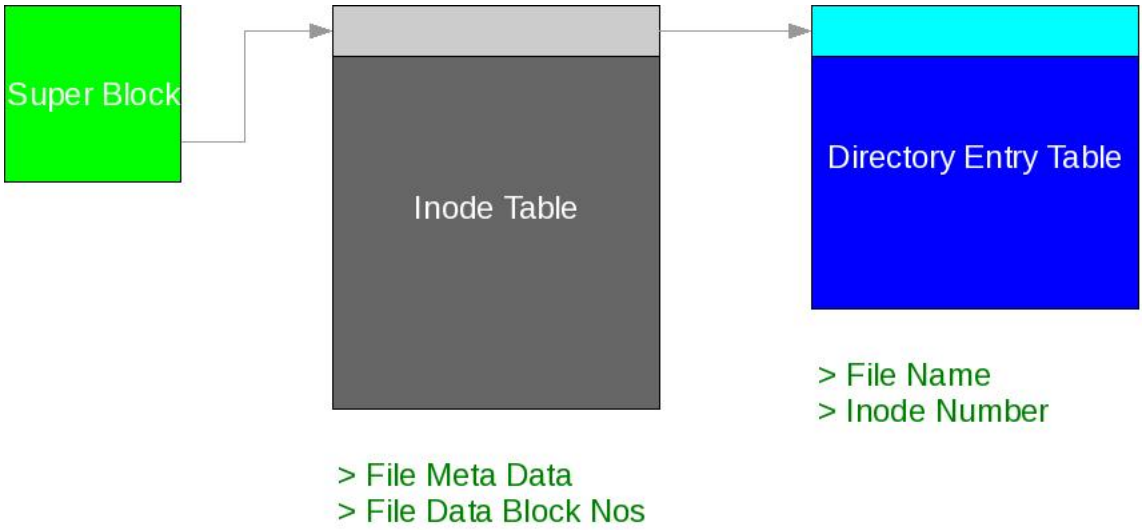
```
<linux/blkdev.h>
typedef void (*request_fn_proc)(struct request_queue *queue);

request_queue_t *q = blk_init_queue(rqf, lock);
blk_cleanup_queue(q);

struct request *req = elv_next_request(q);
blkdev_dequeue_request(req);
elv_requeue_request(q, req);

struct request
    sector_t sector
    unsigned long nr_sectors
    char *buffer
    rq_data_dir(req)
        zero: read from device
        non-zero: write to the device
    ...
```

7.4 The File System Module



Notes:

7.4.1 Registering & Unregistering the File System



Notes:

```
<linux/fs.h>
register_filesystem(file_system_type *)
unregister_filesystem(file_system_type *)
```

7.4.2 Filesystem Operations

Notes:

```
get_sb: (Invoked by mount)
get_sb_single: Fill the Super Block
    Block Size & Bits
    Magic Number
    Super Block Operations
    File System type
    Root Inode
    Inode Operations
    Inode Mode
    Address Operations
    File Operations
kill_sb: (Invoked by umount)
kill_anon_super
```

7.4.3 Super Block Operations

Notes:

```
write_inode: Update File Meta Data
statfs: Get the File Status (Invoked by stat)
    simple_statfs (Handler from libfs)
```

7.4.4 Page Operations

Notes:

```
readpage (Invoked by generic_file_aio_read)
writepage (Invoked by generic_file_aio_write)
commit_write (Invoked by kswapd)
```

Page Functions

```
PageUptodate/Dirty/Writeback/Locked
SetPageUptodate, ClearPageDirty, unlock_page, ...
page_address, ...
```

7.4.5 Inode Operations

Notes:

```
lookup (invoked for file metadata lookup)
```

Inode Information

```
Size
Mode
File Operations
...
```

7.4.6 File Operations

The Usual Ones:

Notes:

```
open -> generic_file_open
release
read -> do_sync_read -> generic_file_aio_read
write -> do_sync_write -> generic_file_aio_write
aio_read -> generic_file_aio_read
aio_write -> generic_file_aio_write
```

The New Ones:

Notes:

```
readdir: Change Directory (cd). List Directory (ls)
    Directory Entry
    Fill Function
fsync: Sync the File (sync)
    simple_sync_file
```

7.5 Exercises:

Chapter 8

PCI Drivers

8.1 Why PCI?

Notes:

8.2 PCI Addressing

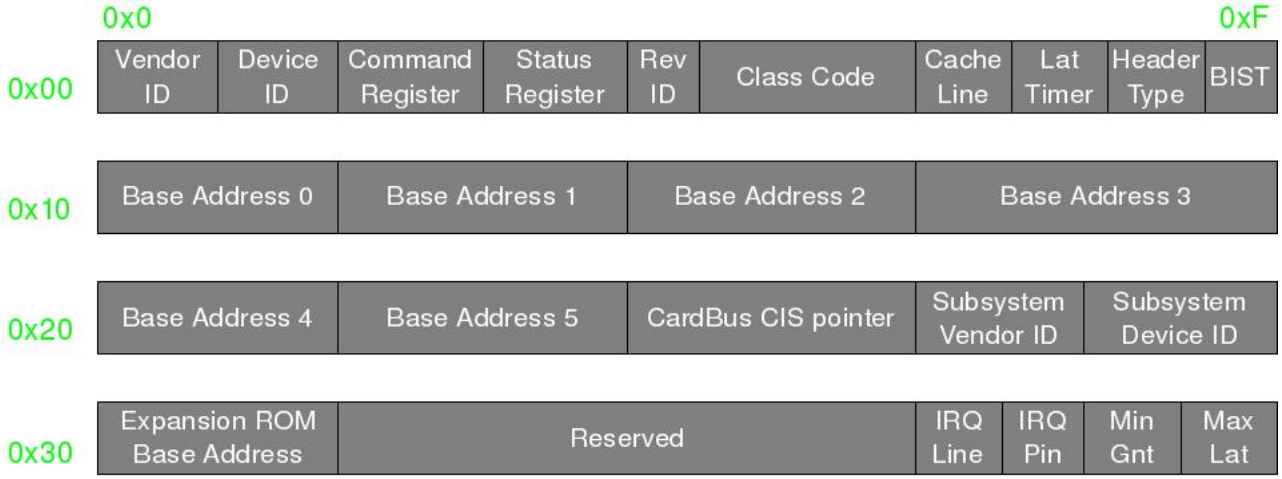
Notes: Buses (upto 256) -> Devices (upto 32) -> Functions (upto 8)

8.2.1 Interface Structure Viewing Options

Notes:

```
lspci
cat /proc/bus/pci/devices
tree /sys/bus/pci/devices
```

8.3 PCI Configuration Space



Notes:

8.3.1 PCI Configuration Space Access APIs

Notes:

```
<linux/pci.h>
pci_read_config_byte/word/dword(struct pci_dev *dev, int where, u8/16/32 *val);
pci_write_config_byte/word/dword(struct pci_dev *dev, int where, u8/16/32 *val);
```

8.4 PCI Driver Registration

Notes:

```
int pci_register_driver(struct pci_driver *drv);
int pci_unregister_driver(struct pci_driver *drv);
struct pci_driver
    const char *name
    const struct pci_dev_id *id_table;
        PCI_DEVICE(vendor, device);
        PCI_DEVICE_CLASS(dev_class, dev_class_mask);
    int (*probe)(pci_dev, id_table);
    void (*remove)(pci_dev);
```

8.4.1 The PCI probe

Notes:

```
int probe(struct pci_dev *d, struct pci_dev_id *id)
{
    /* Initialize the PCI Device */
    ...
    /* Enable the PCI Device */
    pci_enable_device(d);

    ...
    return 0; /* Claimed. Negative for not Claimed */
}
```

8.5 PCI Device Access

Notes:
Upto 6 Memory or I/O regions
BAR: Base Address Register

8.5.1 Operations

```
unsigned long pci_resource_start(dev, bar);
unsigned long pci_resource_end(dev, bar);
unsigned long pci_resource_flags(dev, bar);
<linux/ioport.h>
    flags: IORESOURCE_IO, IORESOURCE_MEM, IORESOURCE_PREFETCH
```