Linux Device Drivers

Deep dive into leveraging devices

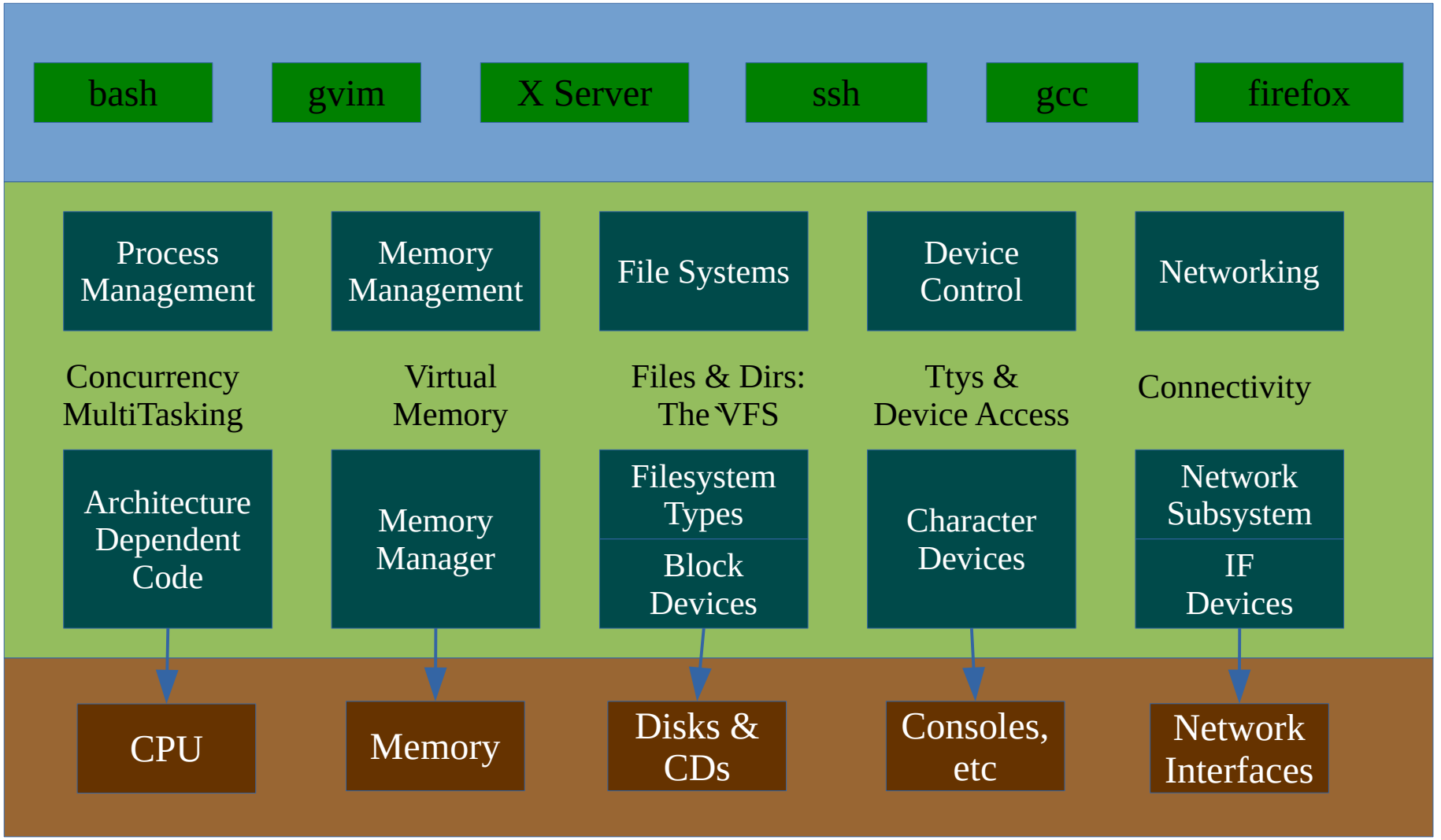
Team Emertxe

Introduction

Familiarity Check

- Good C & Programming Skills
- Linux & the Filesytem
  - Root, User Space Headers & Libraries
- Files
  - Regular, Special, Device
- Toolchain
  - gcc & friends
- Make & Makefiles
- Kernel Sources (Location & Building)

Linux Driver Ecosystem



The Flow

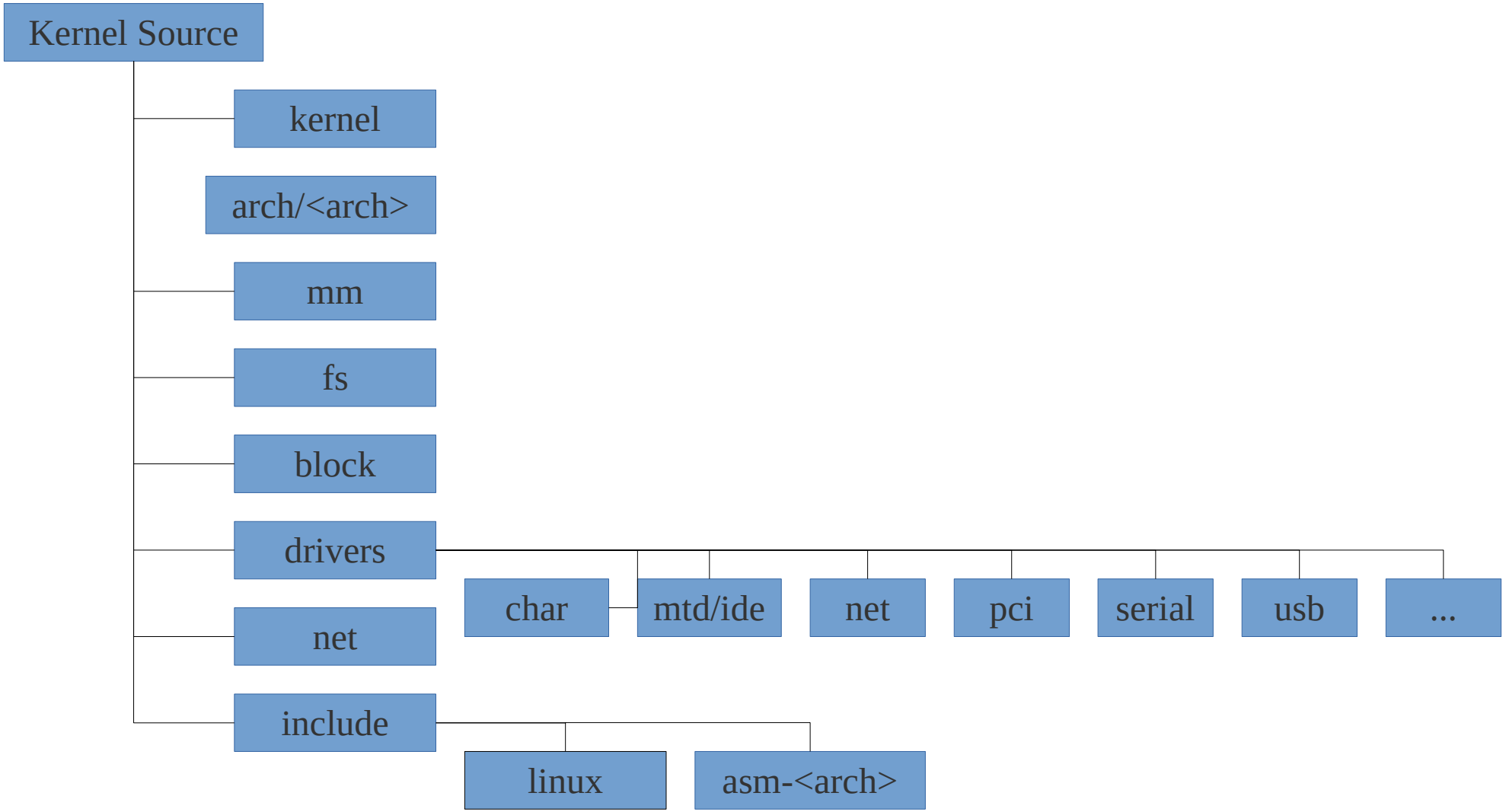
- Introduction
- Character Drivers
- Memory & Hardware
- Time & Timings
- USB Drivers
- Interrupt Handling
- Block Drivers
- PCI Drivers

EMERTXE

Hands-On

- Your First Driver
- Character Drivers
  - Null Driver
  - Memory Driver
  - UART Driver for Customized Hardware
- USB Drivers
  - USB Device Hot-plug-ability
  - USB to Serial Hardware Driver
- Filesystem Modules
  - VFS Interfacing
  - “Pseudo” File System with Memory Files

Kernel Source Organization



The Locations & Config Files

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

The Commands

- lsmod
- insmod
- modprobe
- rmmod
- dmesg
- objdump
- nm
- cat /proc/<file>

The Kernel's C

- ctor & dtor
  - init_module, cleanup_module
- printf
  - printk
- Libraries
  - <kernel src>/kernel
- Headers
  - <kernel src>/include

The Init Code

```
static int __init mfd_init(void)
{
    printk(KERN_INFO "mfd registered");

    ...

    return 0;
}
module_init(mfd_init);
```

The Cleanup Code

```
static void __exit mfd_exit(void)
{
    printk(KERN_INFO "mfd deregistered");

    ...
}
module_exit(mfd_exit);
```

Usage of printk

- <linux/kernel.h>
- Constant String for Log Level
  - 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 */
- printf like arguments

The Other Basics & Ornaments

- Headers
  - #include <linux/module.h>
  - #include <linux/version.h>
  - #include <linux/kernel.h>
- MODULE_LICENSE("GPL");
- MODULE_AUTHOR("Emertxe");
- MODULE_DESCRIPTION("First Device Driver");

Building the Module

- Our driver needs
  - The Kernel Headers for Prototypes
  - The Kernel Functions for Functionality
  - The Kernel Build System & the Makefile for Building
- Two options
  - Building under Kernel Source Tree
    - Put our driver under drivers folder
    - Edit Kconfig(s) & Makefile to include our driver
  - Create our own Makefile to do the right invocation

Our 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
```

Try Out your First Driver

Character Drivers

Major & Minor Number

- ls -l /dev
- Major is to Driver; Minor is to Device
- <linux/types.h> (>= 2.6.0)
  - dev_t: 12 & 20 bits for major & minor
- <linux/kdev_t.h>
  - MAJOR(dev_t dev)
  - MINOR(dev_t dev)
  - MKDEV(int major, int minor)

Registering & Unregistering

- Registering the Device Driver
  - 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);
- Unregistering the Device Driver
  - void unregister_chrdev_region(dev_t first, unsigned int count);
- Header: <linux/fs.h>

The file operations

- #include <linux/fs.h>
- struct file_operations
  - 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 *);
  - struct module owner = THIS_MODULE; / linux/module.h> */
  - loff_t (*llseek)(struct file *, loff_t, int);
  - int (*unlocked_ioctl)(struct file *, unsigned int, unsigned long);

User level I/O

- int open(const char *path, int oflag, ...)
- int close(int fd);
- ssize_t write(int fd, const void *buf, size_t nbyte)
- ssize_t read(int fd, void *buf, size_t nbyte)
- int ioctl(int d, int request, ...)
  - The ioctl() function manipulates the underlying device parameters of special files.
  - The argument d must be an open file descriptor.
  - The second argument is a device-dependent request code.

The file & inode structures

- struct file
  - mode_t f_mode
  - loff_t f_pos
  - unsigned int f_flags
  - struct file_operations *f_op
  - void * private_data
- struct inode
  - unsigned int iminor(struct inode *);
  - unsigned int imajor(struct inode *);

Registering the file operations

- #include <linux/cdev.h>
- 1st way initialization:
  - struct cdev *my_cdev = cdev_alloc();
  - my_cdev->owner = THIS_MODULE;
  - my_cdev->ops = &my_fops;
- 2nd way initialization:
  - struct cdev my_cdev;
  - cdev_init(&my_cdev, &my_fops);
  - my_cdev.owner = THIS_MODULE;
  - my_cdev.ops = &my_fops;

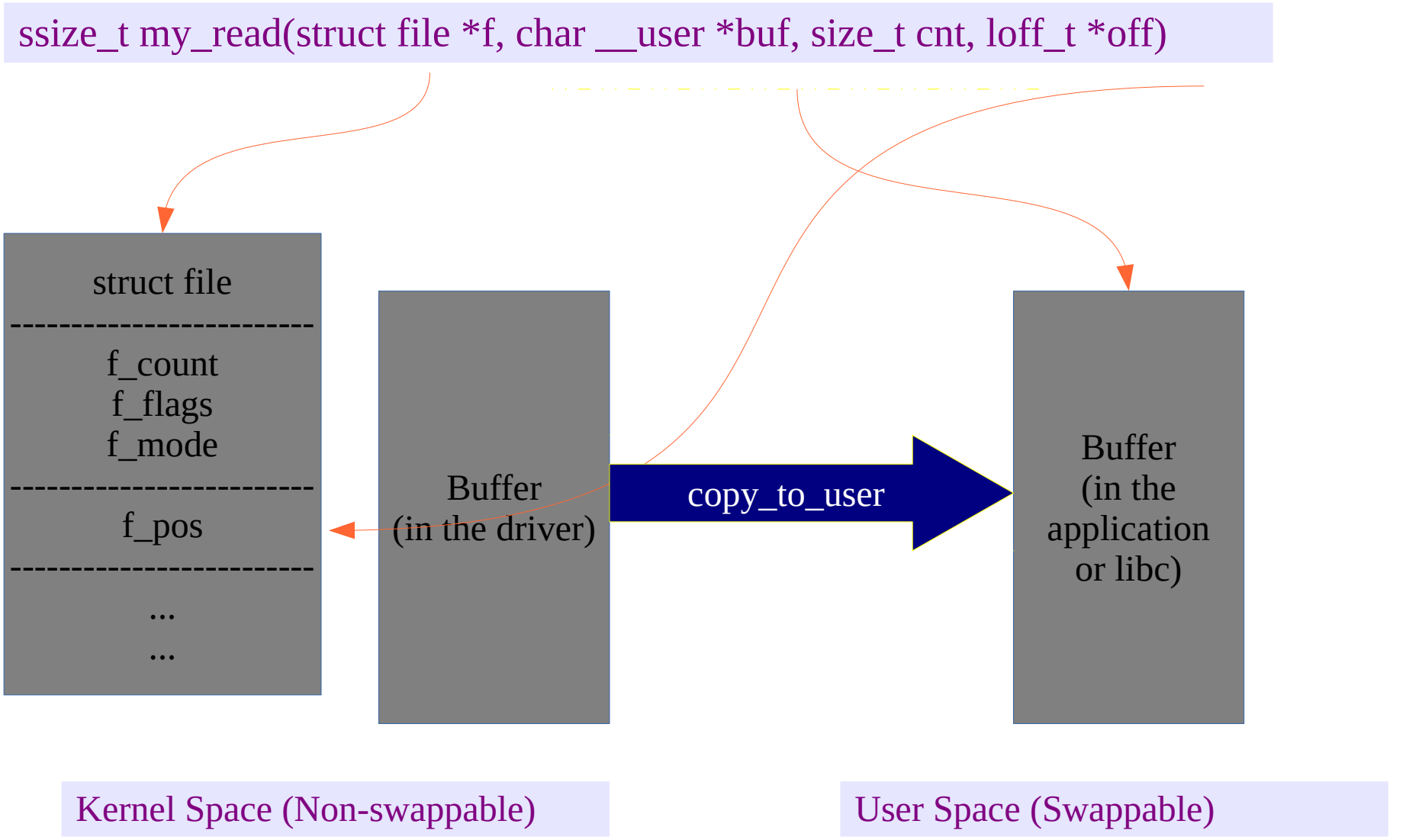
Registering the file operations...

- The Registration
  - int cdev_add(struct cdev *cdev, dev_t num, unsigned int count);
- The Unregistration
  - void cdev_del(struct cdev *cdev);

Registering/Unregistering Old Way

- Registering the Device Driver
  - int register_chrdev(undigned int major, const char *name, struct file_operations *fops);
- Unregistering the Device Driver
  - int unregister_chrdev(undigned int major, const char *name);

The read flow



The /dev/null read & write

```
ssize_t my_read(struct file *f, char __user *buf, size_t cnt, loff_t
    *off)
{
      ...
      return read_cnt;
}
ssize_t my_write(struct file *f, char __user *buf, size_t cnt, loff_t
    *off)
{
      ...
      return wrote_cnt;
}
```

The mem device read

```
ssize_t my_read(struct file *f, char __user *buf, size_t cnt, loff_t
    *off)
{
      ...
      if (copy_to_user(buf, from, cnt) != 0)
      {
            return -EFAULT;
      }
      ...
      return read_cnt;
}
```

The mem device write

```
ssize_t my_write(struct file *f, char __user *buf, size_t cnt, loff_t
    *off)
{
      ...
      if (copy_from_user(to, buf, cnt) != 0)
      {
            return -EFAULT;
      }
      ...
      return wrote_cnt;
}
```

Dynamic Device Node & Classes

- Class Operations
  - struct class *class_create(struct module *owner, char *name);
  - void class_destroy(struct class *cl);
- Device into & Out of Class
  - struct class_device *device_create(struct class *cl, NULL, dev_t devnum, NULL, const char *fmt, ...);
  - void device_destroy(struct class *cl, dev_t devnum);

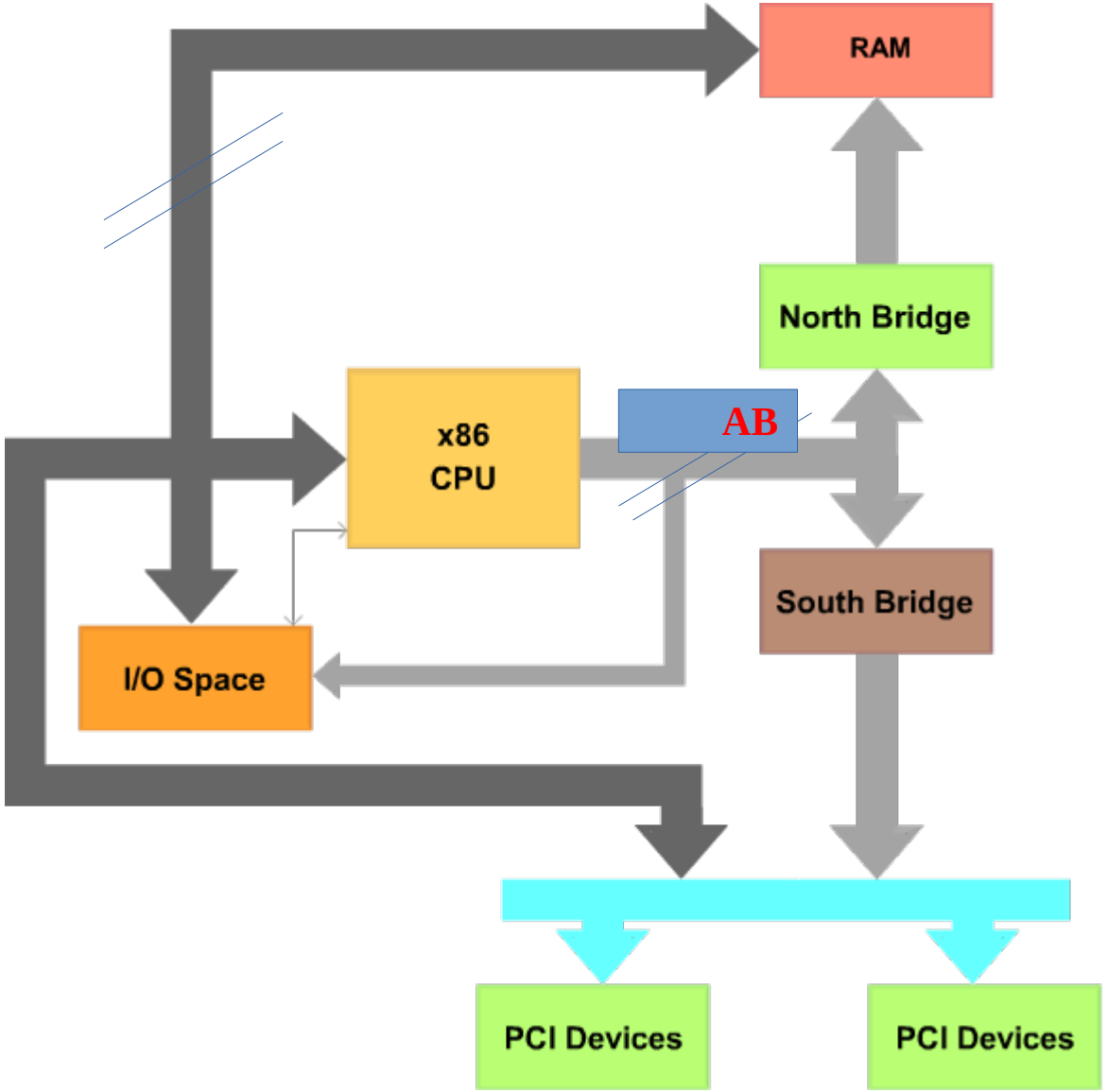
The I/O Control API

- int (*ioctl)(struct inode *, struct file *, unsigned int cmd, unsigned long arg)
- Command
  - <linux/ioctl.h> -> ... -> <asm-generic/ioctl.h>
  - Macros
    - _IO, _IOR, _IOW, _IOWR
  - Parameters
    - type (character) [15:8]
    - number (index) [7:0]
    - size (param type) [29:16]

Module Parameters

- <linux/moduleparam.h>
  - Macros
    - module_param(name, type, perm)
    - module_param_array(name, type, num, perm)
    - Perm (is a bitmask)
      - 0
      - S_IRUGO
      - S_IWUSR | S_IRUGO
  - Loading
    - insmod driver.ko name=10

x86 Architecture

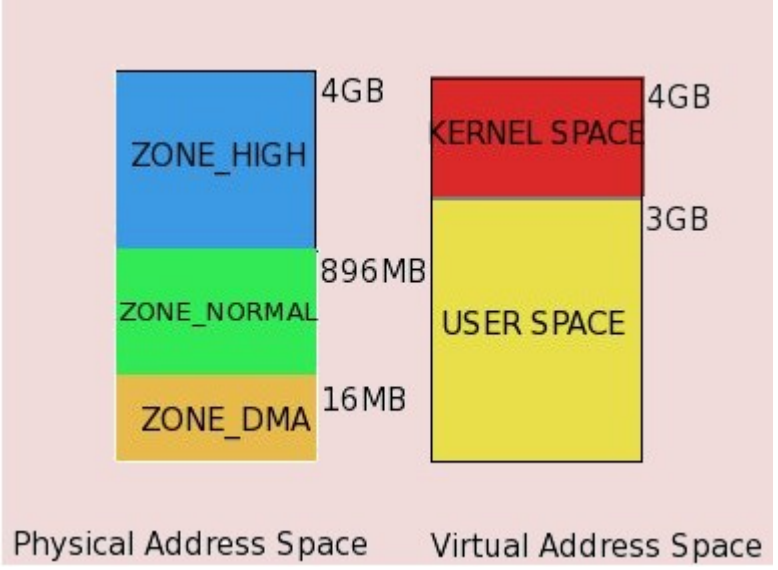


EMERTXE

Memory Access

Physical Vs Virtual Memory

- The kernel Organizes Physical memory in to pages
  - Page size Depends on Arch
    - X86-based 4096 bytes
- On 32-bit X86 system Kernel total Virtual address space
  - Total 4GB (pointer size)
  - Kernel Configuration Splits 4GB in to
    - 3BG Virtual Sp for US
    - 1GB Virtual Sp for Kernel
      - 128MB KDS
  - Virtual Address also called



Memory Access from Kernel Space

- Virtual Address on Physical Address
  - #include <linux/gfp.h>
    - unsigned long __get_free_pages(flags, order); etc
    - void free_pages(addr, order); etc
  - #include <linux/slab.h>
    - void *kmalloc(size_t size, gfp_t flags);
      - GFP_ATOMIC, GFP_KERNEL, GFP_DMA
    - void kfree(void *obj);
  - #include <linux/vmalloc.h>
    - void *vmalloc(unsigned long size);
    - void vfree(void *addr);

Memory Access from Kernel Space...

- Virtual Address for Bus/IO Address
  - #include <asm/io.h>
    - void *ioremap(unsigned long offset, unsigned long size);
    - void iounmap(void *addr);
- I/O Memory Access
  - #include <asm/io.h>
    - unsigned int ioread[8|16|32](void *addr);
    - unsigned int iowrite[8|16|32](u[8|16|32] value, void *addr);
- Barriers
  - #include <linux/kernel.h>: void barrier(void);
  - #include <asm/system.h>: void [r|w|]mb(void);

Hardware Access

I/O Accesses from Kernel Space

- I/O Port Access
  - #include <asm/io.h>
    - unsigned in[b|w|l](unsigned port);
    - void out[b|w|l](unsigned [char|short|int] value, unsigned port);

Hands-On the Hardware

Debugging

Debugging Options

- Printing & syslogd
- Querying
  - /proc
  - Using ioctl
  - sysfs
- Watching
  - strace
  - Oops

Creating /proc entry's

- Creating read-only /proc entry's
  - #include <linux/pros_fs.h>
  - proc_dir_entry *proc_create(const char *name, umode_t mode, struct proc_dir_entry *parent, const struct file_operations *proc_fops);
  - remove_proc_entry(const char *name, struct proc_dir_entry *parent)

Advanced Debugging

- Debuggers
  - gdb <kernel src>/vmlinux /proc/kcore
  - Non-official Kernel Debugger kdb (oss.sgi.com)
  - kgdb & Remote Debugging
- Use Mode Linux (UML)
- LTT (http://www.opersys.com/LTT)

Concurrency

Concurrency & Locking

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

Concurrency without Locking

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

Time Keeping

Time since Bootup

- <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

Absolute Time

- <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);

Delays

Busy Wait & Yielding

- 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_event_timeout
  - wait_event_interruptible_timeout

The WaitQ Example

```
wait_queue_head_t wait;


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

More Precise Busy Waiting

- #include <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);

Back to Yielding

- #include <linux/delay.h>
- void msleep(unsigned int millisecs);
- unsigned long msleep_interruptible(unsigned int millisecs);
- void ssleep(unsigned int secs);

Timers

Kernel Timers

- Back end of the various delays
- <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 *);

Tasklets

- Timers without specific Timing
- <linux/interrupt.h>
- void tasklet_init(struct tasklet_struct *t, void (*func) (unsigned long), unsigned long data);
- DECLARE_TASKLET(name, func, data);
- tasklet_disable/enable(t);
- tasklet_[hi_|]_schedule(t);
- tasklet_kill(t);

Work Queues

In context of “Special Kernel Process”

<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);

Context Specific Functions

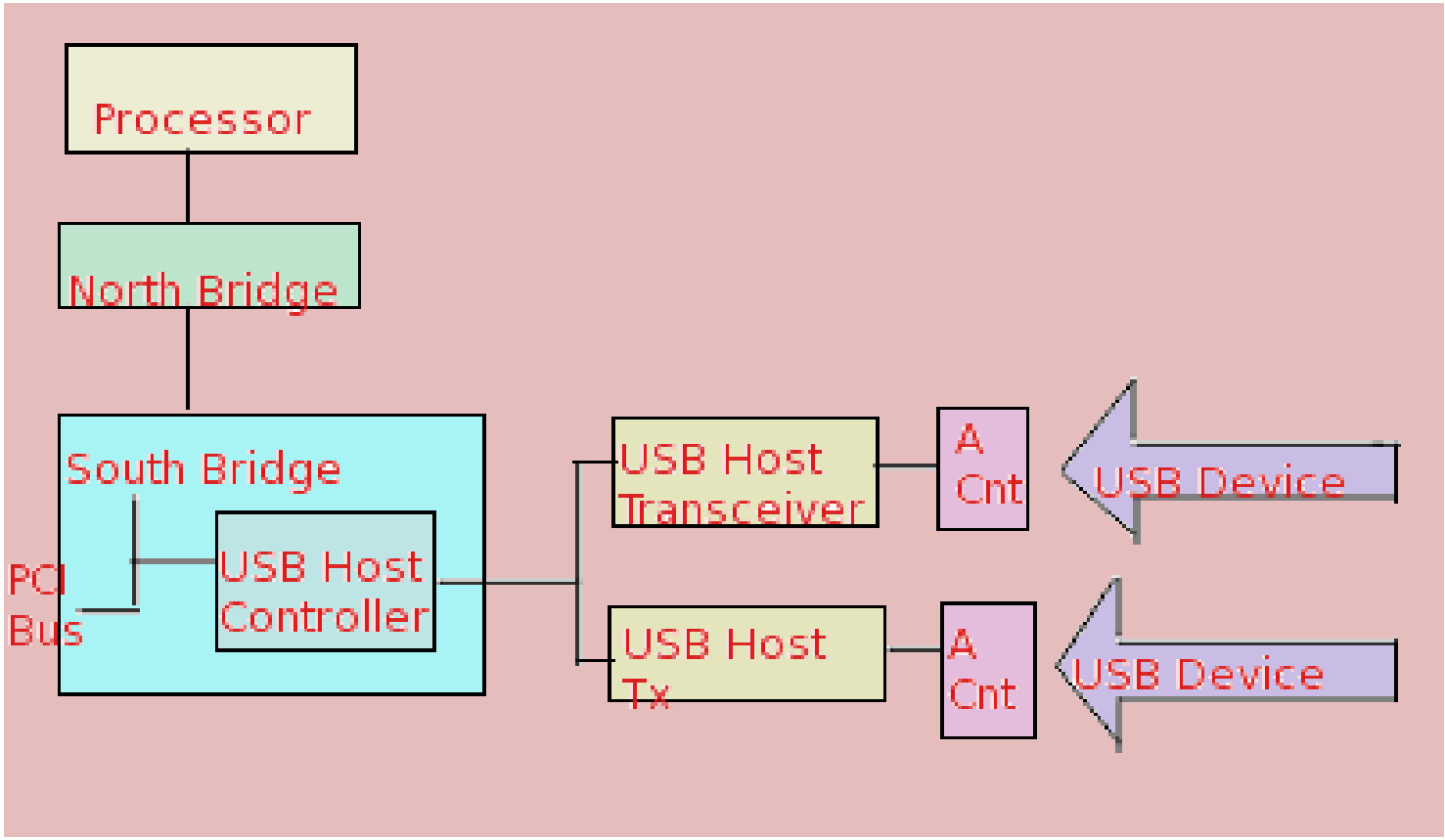
- <asm/hardirq.h>
- in_interrupt()
- in_atomic()

USB Drivers

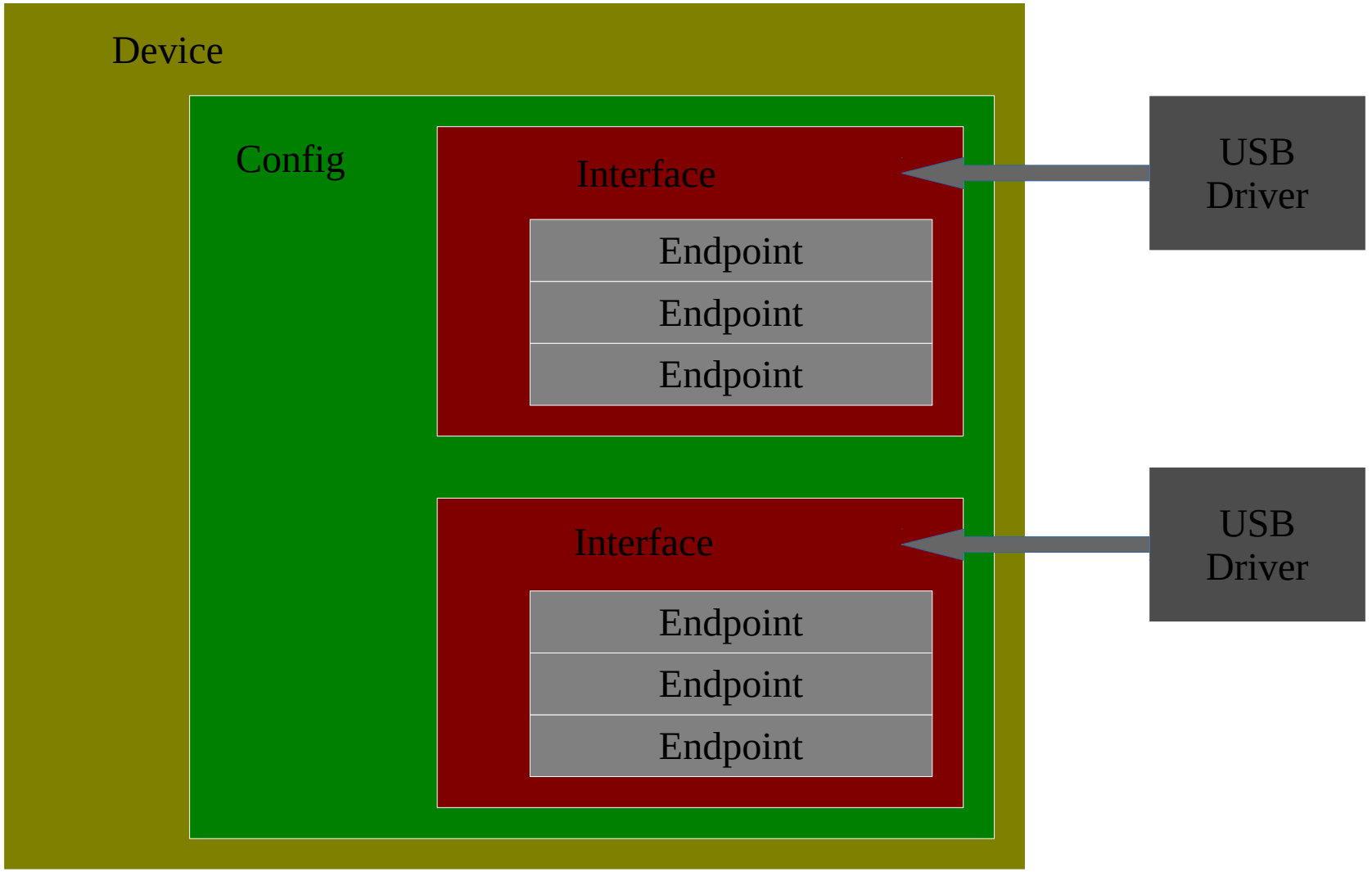
USB Key Features

- Single Master on Host
- Polls the Peripheral Slaves
- Started as a Unified Bus for Slow Devices
- High Speed Specifications
- Bandwidth Allocation Ability
- Linux Device Driver Support
  - USB Host Driver
  - USB Gadget Driver

USB In PC Env


Processor
North Bridge
South Bridge
PCI
Bus
USB Host
Controller
USB Host
Transceiver
A
Cnt
USB Device
USB Host
Tx
A
Cnt
USB Device

USB Device Overview



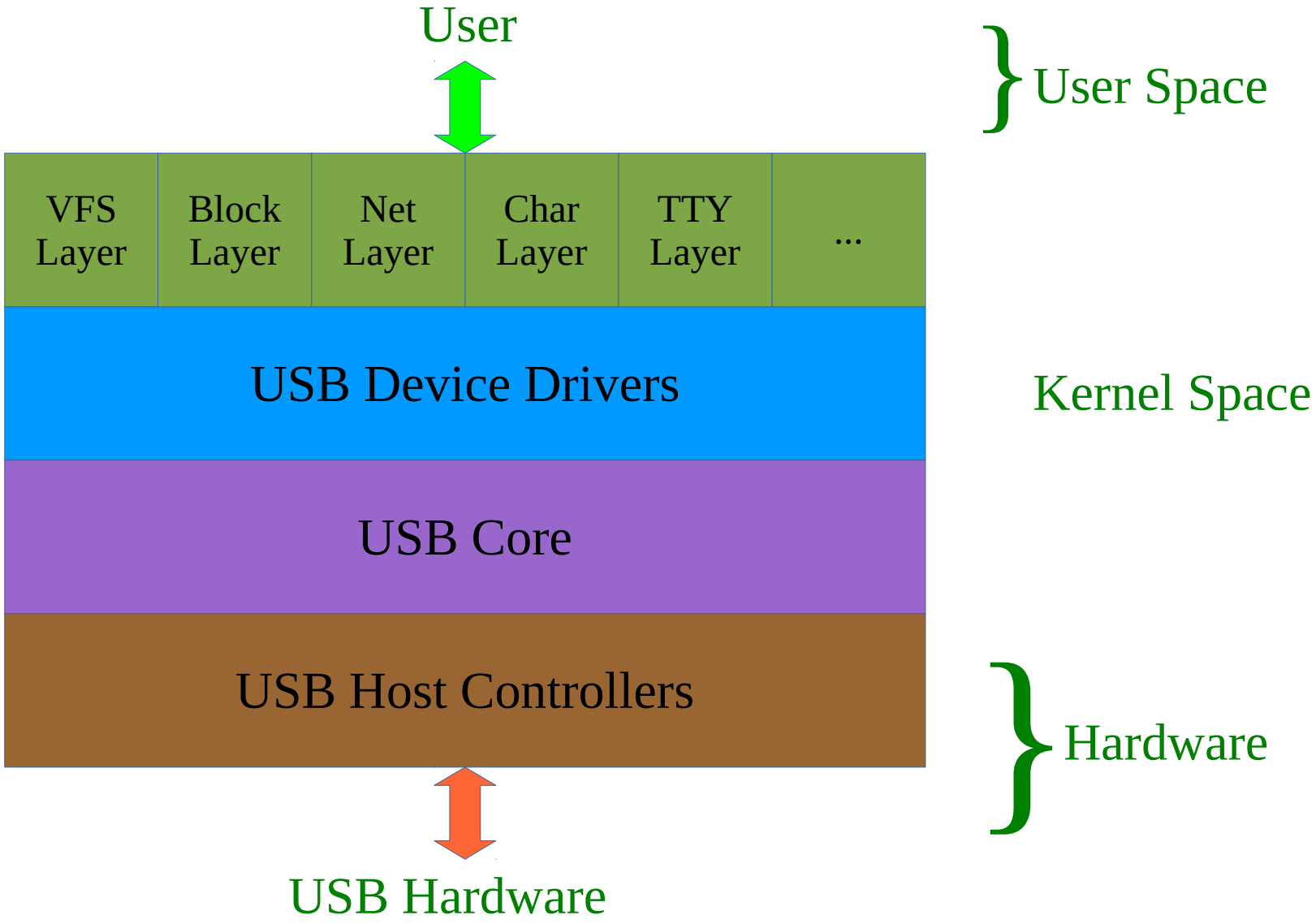
USB Endpoints

- Also called Pipes
- Direction
  - OUT (host->device)
  - IN (device->host)
- Types
  - Control
  - Interrupt
  - Bulk
  - Isochronous

USB Data Structures

- struct usb_host_endpoint
  - struct usb_endpoint_descriptor
- struct usb_interface
- struct usb_host_config
- struct usb_device
  - interface_to_usbdev

USB Driver Overview



USB Core Driver

- Sysfs
- USB Driver Registration / Unregistration
- USB Device Hot Plugability
  - probe: USB Device Registration
  - disconnect: USB Device Unregistration
- USB Transfers through URBs

USB & Sysfs

- /sbin/lspci
  - <dom>:<bus>:<dev>:<fn> for <usbhubid>
- /sys/devices/pci0000:00/<usbhubid>/usb<hub>
  - usb_device fields
  - roothub-hubport:config.interface
    - usb_interface fields

USB Driver Registration

- 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 *)

USB Device Hot-plug-ability

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

USB Request Block

- struct urb
  - 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

URB Operations

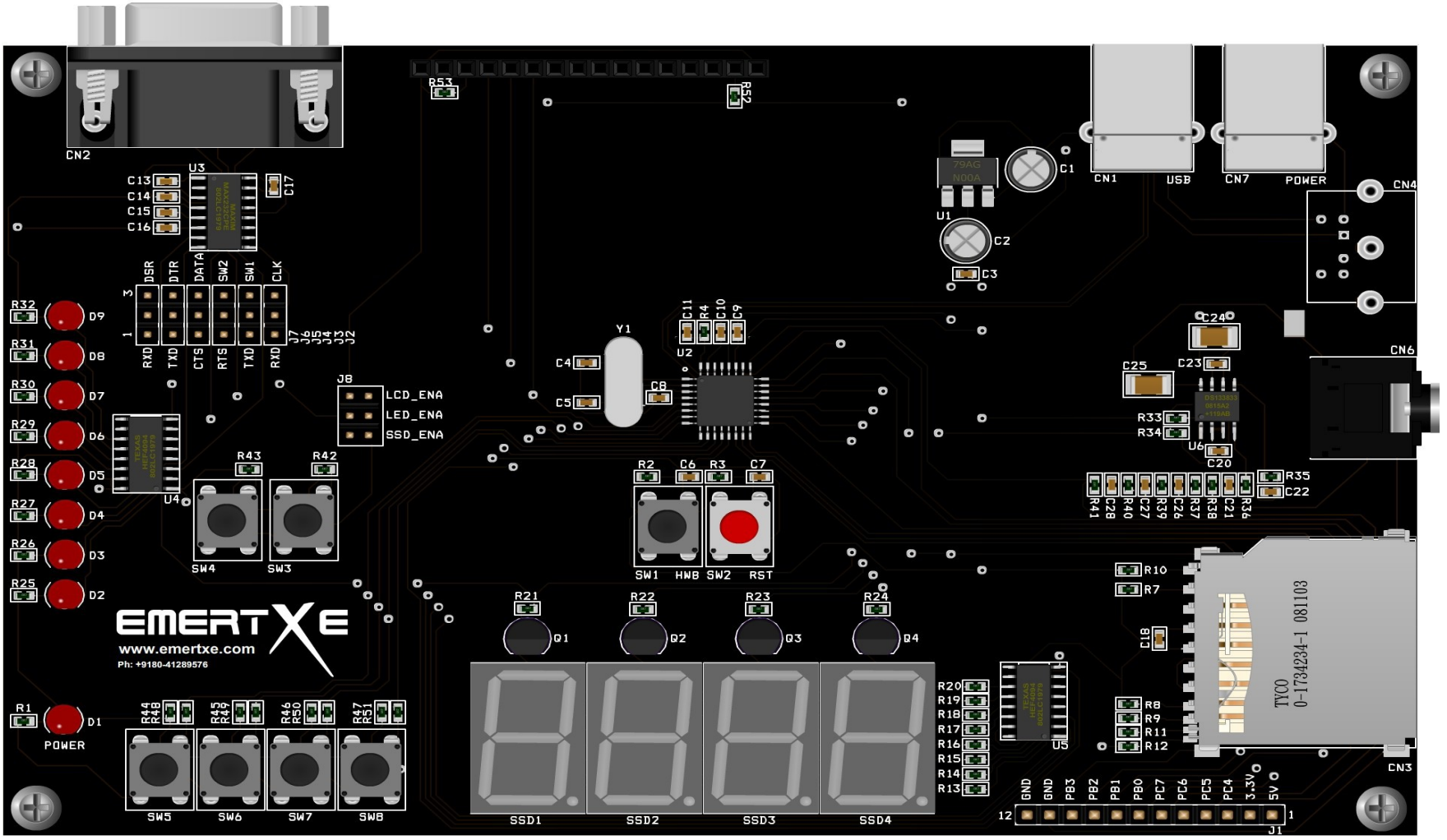
- usb_alloc_urb / usb_free_urb
- usb_fill_[int|bulk|control]_urb
- usb_submit_urb
- usb_unlink_urb / usb_kill_urb

USB Transfer Wrappers

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

The Real USB Device

USB to Serial Hardware



EMERTXE

Interrupts

LSPs

- Architecture Specific Code:
  - arch/<arch>
  - include/asm
- Machine Specific Code:
  - arch/<arch>/mach-*
  - include/asm/mach-*
- Linux Support Package
  - Machine or Board Specific Code
  - include/asm/mach-default

IRQs

- Numbers derived by CPU & Board
- Programmed in Interrupt Controller
- x86 Specific: 0x00 to 0x1F
- Board Specific: 0x20 to 0xFF
  - <asm/interrupt.h> -> <asm/irq.h> -> irq_vectors.h

The APIs

<asm/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_TRIGGER_RISING , IRQF_TRIGGER_FALLING, IRQF_SHARED, ...

The Additional Info

- IRQ Control
  - enable_irq(irq)
  - disable_irq(irq)
- Autoprobing IRQs
  - irqs = probe_irq_on();
  - irq = probe_irq_off(irqs);
- IRQ Handler returns
  - IRQ_NONE
  - IRQ_HANDLED

Soft IRQs

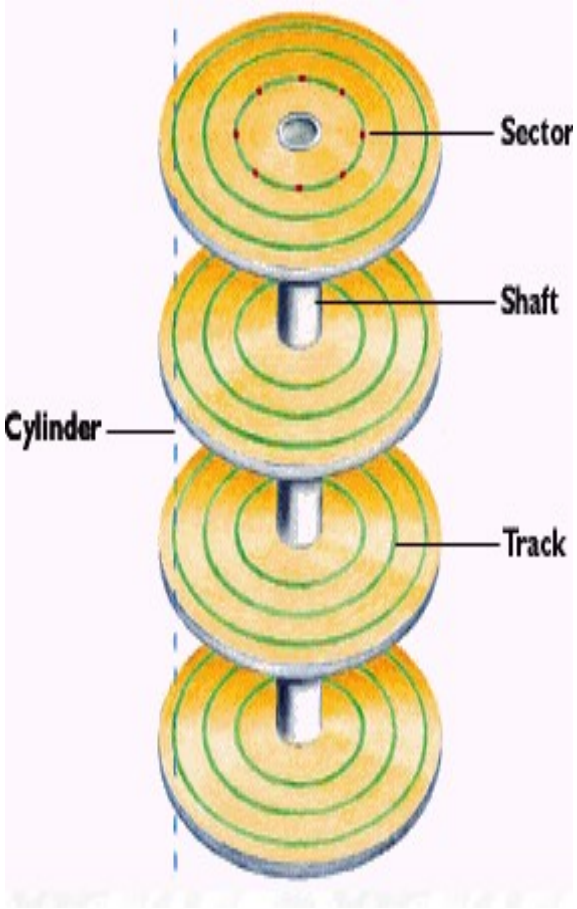
- Timer
- Network
- Block
- Tasklet
- Scheduler
- High Resolution Timer

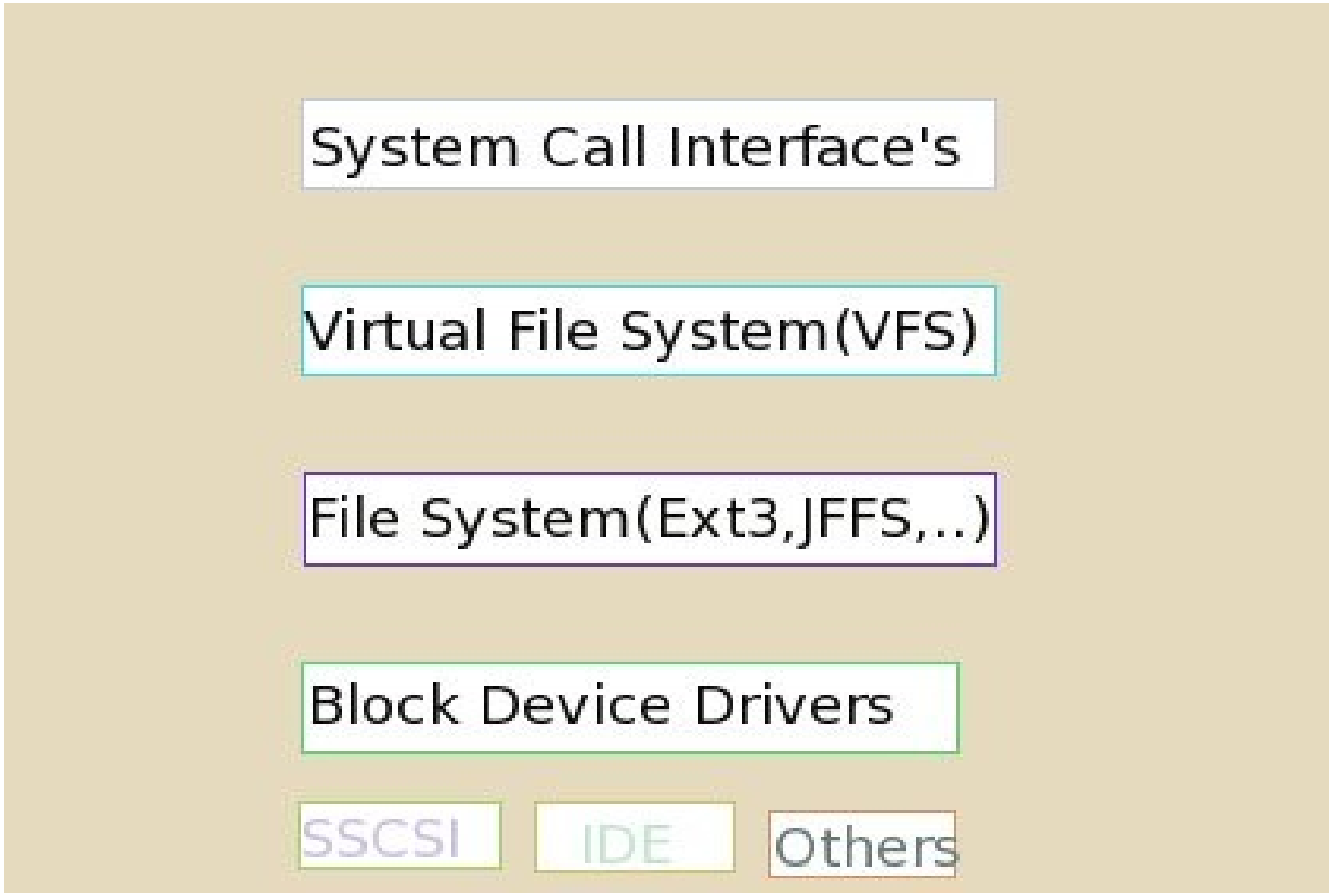
Top & Bottom Halves

- Top Half
  - Registered through request_irq
- Bottom Half
  - Tasklet
    - Soft IRQ Context
    - Fast & Atomic
  - Workqueue
    - Special Kernel Process Context
    - Allowed to sleep

Block Drivers

How the Block Device fits in Linux Kernel





EMERTXE

Registration

- Driver Registration
  - <linux/fs.h>
  - int register_blkdev(major, name);
  - int unregister_blkdev(major, name);
- Disk Drive Registration
  - <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

Block Device Operations

- int open(struct block_device *, fmode_t);
- int close(struct gendisk *, fmode_t);
- int ioctl(struct block_device *, fmode_t, unsigned, unsigned long);
- int compat_ioctl(struct block_device *, fmode_t, unsigned, unsigned long);
- int media_changed(struct gendisk *);
- int revalidate_disk(struct gendisk *gd);
- int getgeo(struct block_device *, struct hd_geometry *);
- struct module *owner;

Request Queues & Processing

<linux/blkdev.h>

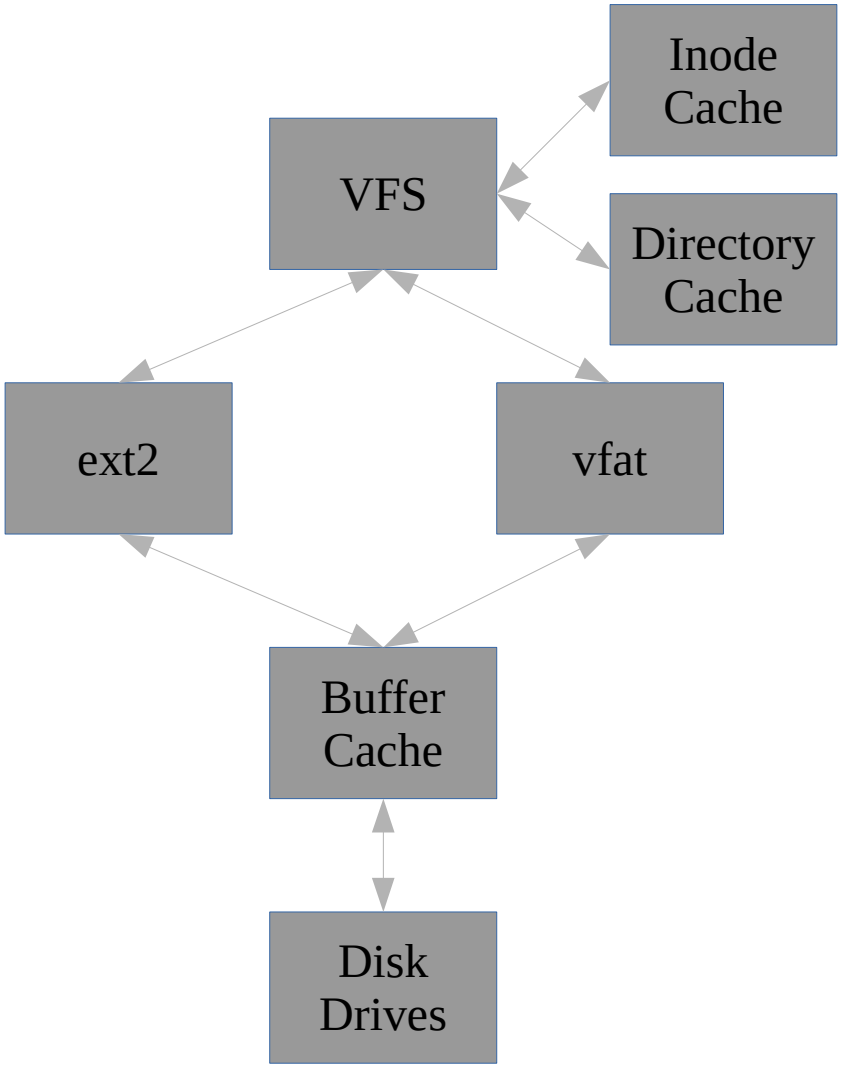
- typedef void (*request_fn_proc)(request_queue_t *queue);
- request_queue_t *q = blk_init_queue(rqf, lock);
- blk_cleanup_queue(q);
- struct request *req = blk_fetch_request(q);
- __blk_end_request(req);
- __blk_end_request_all(req, ret);

struct request

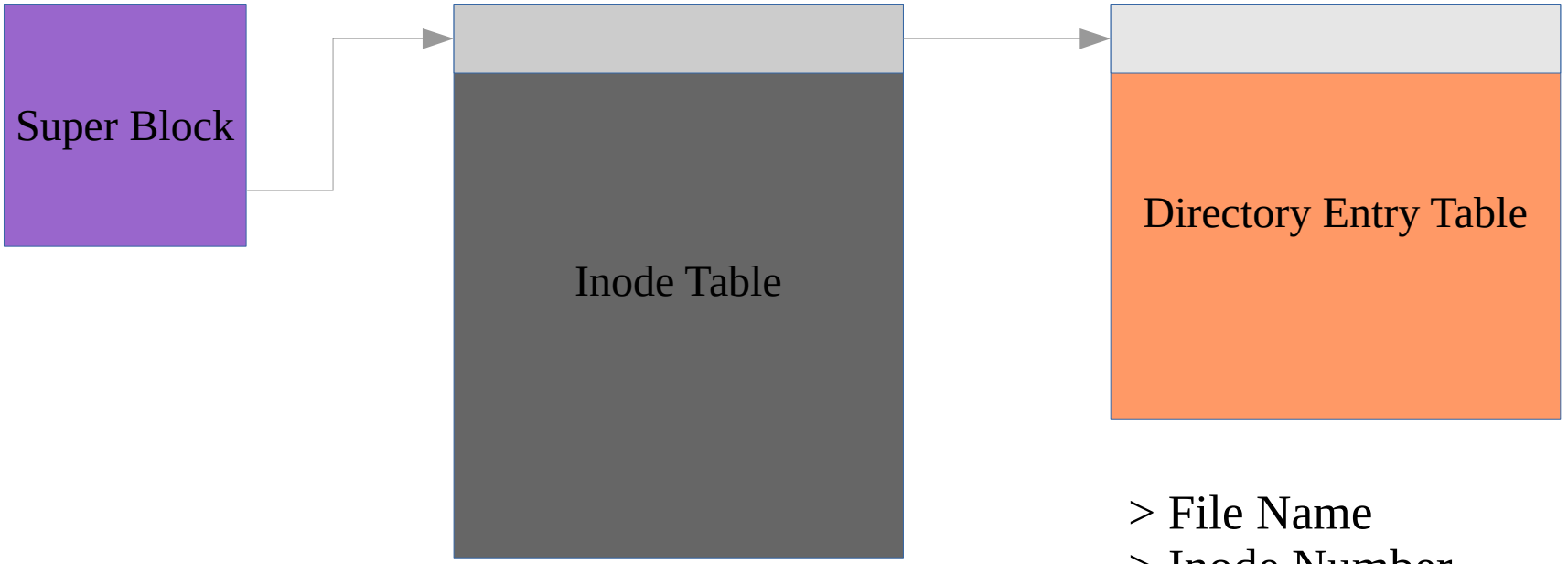
- sector_t sector
- unsigned long nr_sectors
- char *buffer
- rq_data_dir(req)
  - zero: read from device
  - non-zero: write to the device
- ...

FileSystem Modules

Virtual File System Interfaces



Virtual File System Internals



> File Meta Data
> File Data Block Nos

> File Name
> Inode Number

Registration

- <linux/fs.h>
- register_filesystem(file_system_type *)
- unregister_filesystem(file_system_type *)
- Filesystem Operations
  - mount: (Invoked by mount)
    - fill_super -> Fill the Super Block
  - kill_sb: (Invoked by umount)

Fill the Super Block

- Block Size & Bits
- Magic Number
- Super Block Operations
- File System type
- Root Inode
  - Inode Operations
  - Inode Mode
  - File Operations

Super Block Operations

- read_inode: Get the Address Operations
- write_inode: Update File Meta Data
- statfs: Get the File Status (Invoked by stat)
  - simple_statfs (Handler from libfs)

Address Space Operations

- Page Cache
- Page Operations
  - readpage (Invoked by generic_file_read)
  - write_start
  - writepage (Invoked by generic_file_write)
  - write_end
- Page Functions
  - PageUptodate/Dirty/Writeback/Locked
  - SetPageUptodate, ClearPageDirty, unlock_page, ...
  - page_address, ...

Inode Operations

- lookup
  - Search the file name by absolute path
  - Traverse the inode chain
  - if found matching inode, then populate the dentry with the inode information
- Inode Information
  - Size
  - Mode
  - File Operations
  - ...

File Operations

- The Usual Ones
  - open -> generic_file_open
  - read -> generic_file_read
  - write -> generic_file_write
  - release
- The New Ones
  - readdir: Change Directory (cd). List Directory (ls)
    - Directory Entry
    - Fill Function
  - fsync: Sync the File (sync)
    - simple_sync_file

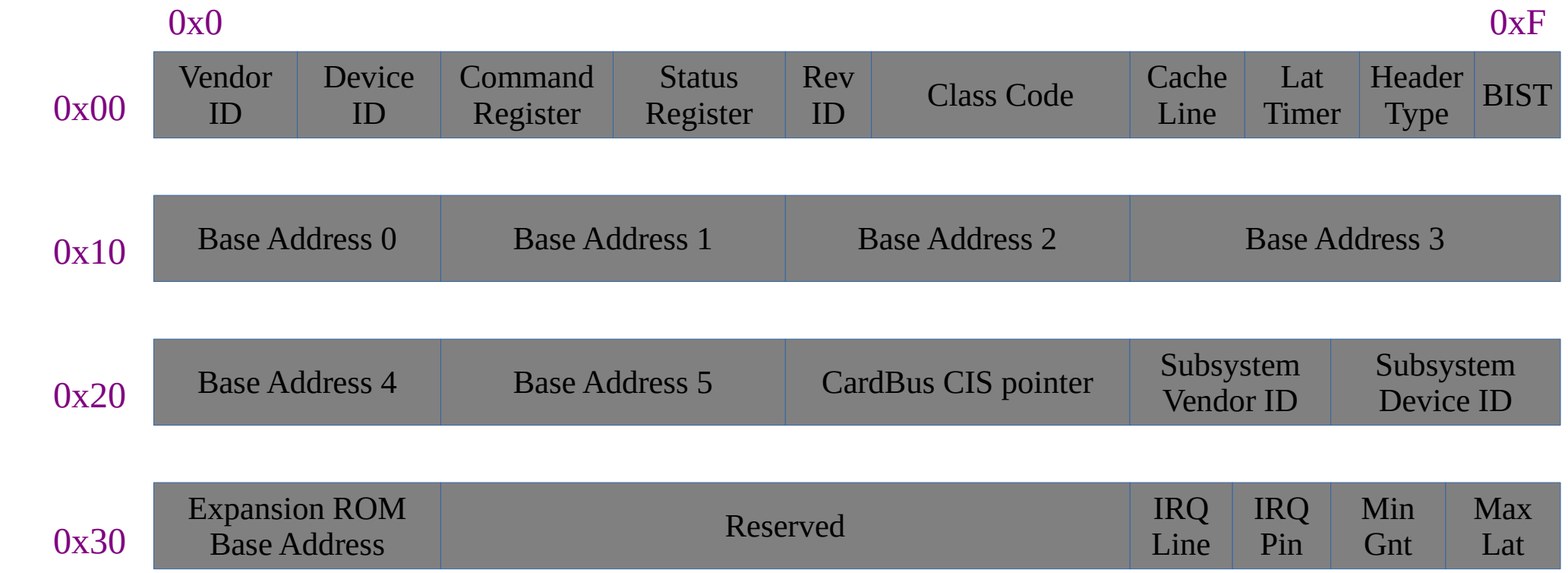
Experiments with our File System

PCI Drivers

PCI Interface

- Replacement for ISA
  - Fast & Platform independent
- Addressing
  - Buses (upto 256)
    - Devices (upto 32)
      - Functions (Upto 8)
- Viewing the Interface Structure
  - lspci
  - cat /proc/bus/pci/devices
  - tree /sys/bus/pci/devices

PCI Device Configuration Space



EMERTXE

PCI Configuration Space...

- During Bootup
  - By BIOS
  - By PCI Core (Bootloader or Kernel)
- <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);

PCI Driver Registration

- 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);

The 'probe' Function

```
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 */

}
```

EMERTXE

Old-style PCI Probing & Getting

- struct pci_dev *pci_find_*(vendor, device, from);
- struct pci_dev *pci_get_device(v, d, from);
- struct pci_dev *pci_get_subsys(v, d, ssv, ssd, f);
- struct pci_dev *pci_get_slot(bus, devfn);
- pci_dev_put(pci_dev);

PCI Device Access & Operations

- Upto 6 Memory or I/O regions
- unsigned long pci_resource_start(dev, bar);
- unsigned long pci_resource_end(dev, bar);
- unsigned long pci_resource_flags(dev, bar);
  - <linux/ioport.h>
  - IORESOURCE_IO
  - IORESOURCE_MEM

References

- Kernel 3.x Source
- LDD Slides
  - http://www.slideshare.net/EmertxeSlides
- Linux Device Drivers (3rd Edition) by
  - Jonathan Corbet
  - Alessandro Rubini
  - Greg Kroah-Hartman
- Email Address: mubeenj@emertxe.com

EMERTXE

Feedback Time

Test Yourself

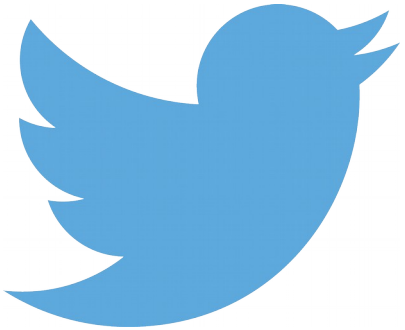
Stay connected

About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Branch Office:
Emertxe Information Technologies,
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046

Corporate Headquarters:
Emertxe Information Technologies,
83, Farah Towers, 1st Floor,
MG Road,
Bangalore, Karnataka - 560001
T: +91 809 555 7333 (M), +91 80 41289576 (L)
E: training@emertxe.com

https://www.facebook.com/Emertxe



https://twitter.com/EmertxeTweet

https://www.slideshare.net/EmertxeSlides

EMERTXE

Thank You