

# Workspace for 'linux-introduction'

Page 11 (row 3, column 5)

{vhi, hi, med, lo, vlo} = scale  
↳ avg

1 Career

assume that we start w/ BSFCE

Yrs of educ. affew BS	Career name	Start & approx	Stress	Satisf-action	Stability	Rate of promot.	Job scar-city	People Centric	Service
0	Engr. BS	75	med	lo -	med	med-	lo	lo	lo
2	Engr. MS	100	med+	med -	med+	med	lo+	lo	lo
4-6	Engr. PhD	115 big upside - more than industry salary	med+ up to you, high at first	hi	hi	med+	med	lo	lo
4-6 experience PhD/MS help	prof. PhD startup	110 more than industry salary	med.	vhi	hi (esp. after tenure)	lo	hi	hi	hi
0	Govt job	65?	lo	vhi	vhi	med-	lo	med	vhi
3 JD	Patent law	Assoc - 150 Partner - 200	vhi vhi	hi hi	med med	lo lo	med med	med hi	lo+ lo+
4? 3? special ization	MD	80	depends on branch	vhi	hi	lo	med	vhi	vhi
2? experience valued experience/ PhD/MS helps	MBA ECE Consultant	150 500K <sup>①</sup>	hi hi	med+ med+	lo+	hi	med- med+	vhi hi	lo hi
		↑ <u>YMMV</u>							

▲ = "someone" pays for tuition etc  
& gives you a monthly salary

Certifications → PE  
Security clearance →

## 2 Career

- Where MS/PhD
- ✓ Prof tenure → 7yr Assistant Prof [evaluated] → tenured (Assoc)  
7+ Associate Prof → not tenured (terminal appt)  
n+ Prof.
- ✓ Startups - VC
- ✓ Consultant
- ✓ Patent law - articulate
  - logical

- 
- Responsive
  - Responsible
  - Respectful
  - thorough

# ECEN 449 – Microprocessor System Design



## Introduction to Linux

not open  
source

open source

↓

operating  
system (OS)

Android  
MS windows  
iOS  
Mac OS

## Objectives of this Lecture Unit

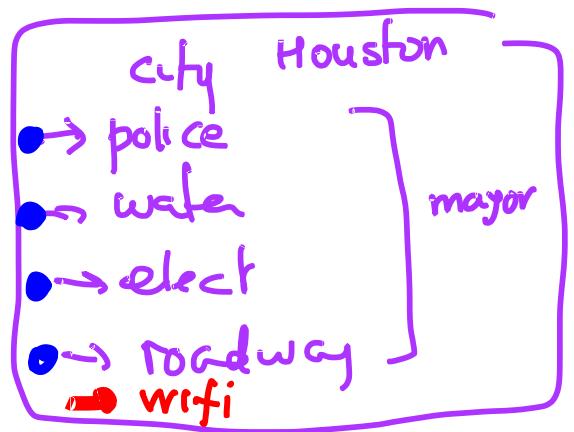
- Learn basics of Linux O/S

## Objectives

- The Operating System environment and services
- How are services accessed in Linux?

## Motivation

- Applications need an execution environment:
  - Portability, standard interfaces
  - File and device controlled access
  - Preemptive multitasking / scheduling
  - Virtual memory (protected memory, paging)  $\sim 4\text{ KB}$
  - Shared libraries
  - Shared copy-on-write executables
  - TCP/IP networking
  - SMP support (symmetric multi processor)
- Hardware developers need to integrate new devices
  - Standard framework to write device drivers
  - Layered architecture dependent and independent code
  - Extensibility, dynamic kernel modules

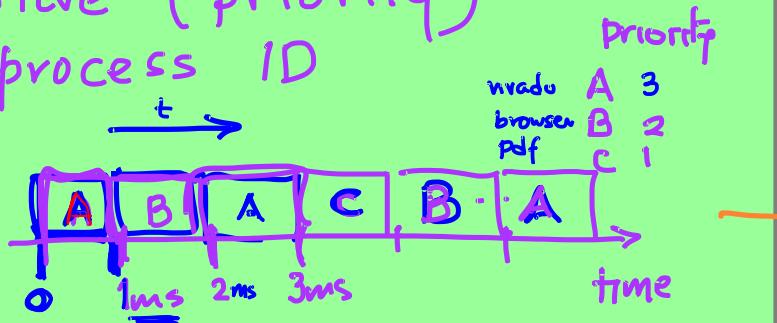


One program (task) @ a time 4

→ multitask

- needs scheduling [ quantum: 1ms ]
- preemptive (priority)
- PID = process ID

{ sched ulng } { round robin  
priority }

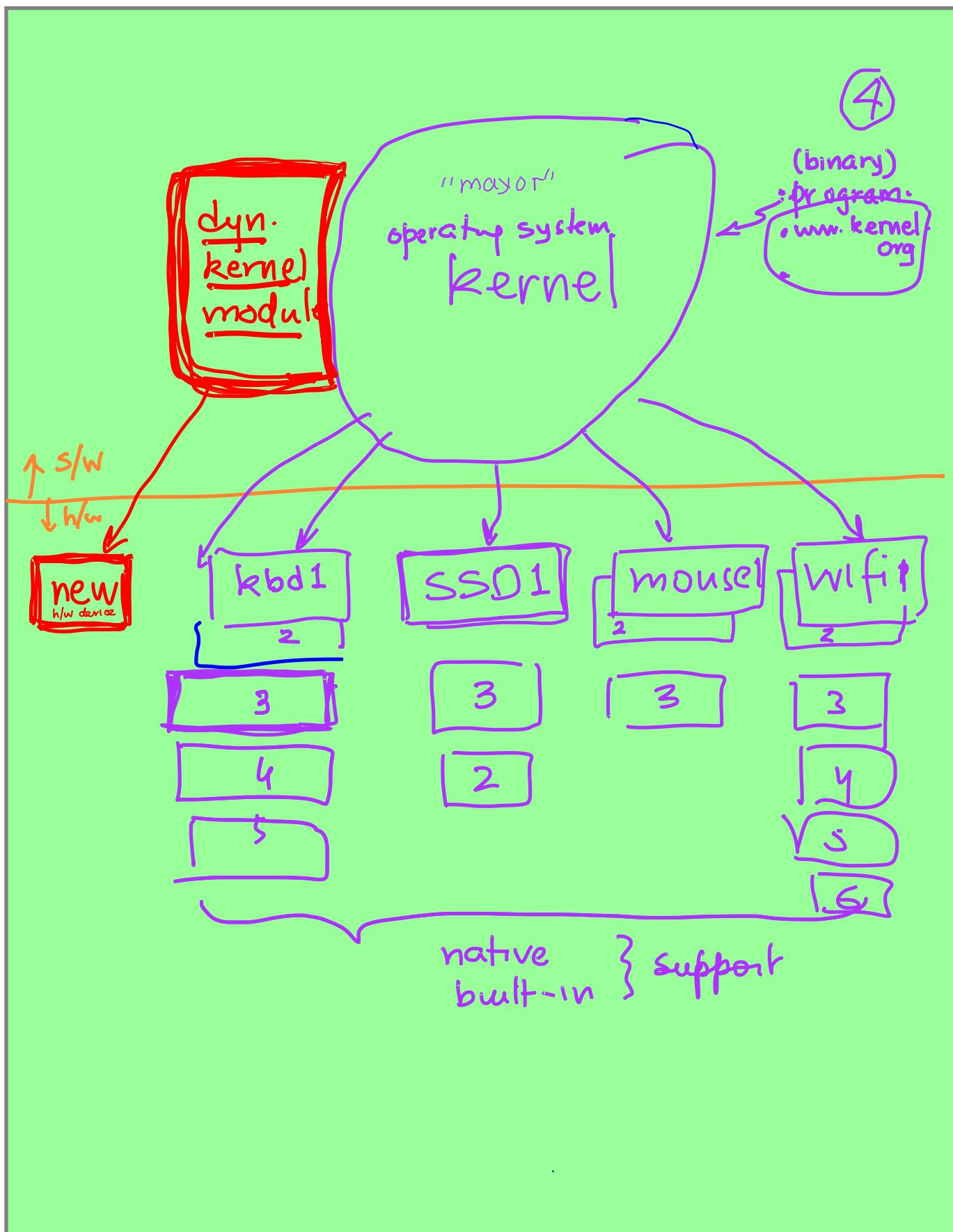


Multi-core w/ k cores

- k tasks @ a time
- still needs scheduling, multitasking, preemption

$$\text{Quant} \quad 10^{-3} \text{ s} \quad f = 1 \text{ GHz} ; \quad T = \frac{1}{f} = 10^{-9} \text{ s}$$

$$\# \text{ instr/quant} = \frac{10^{-3}}{T} = \frac{10^{-3}}{10^{-9}} \simeq 10^6$$



# Linux O/S Architecture

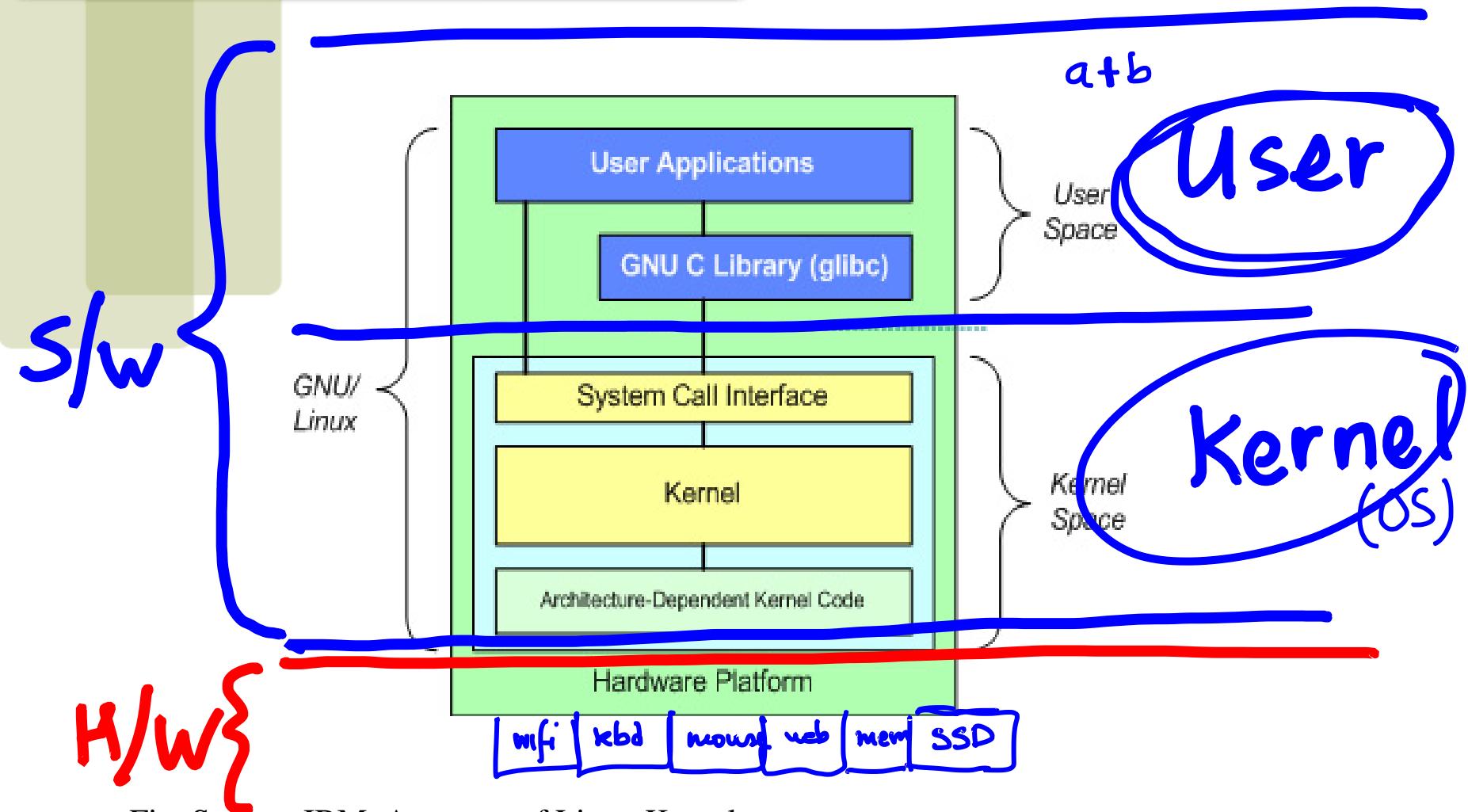
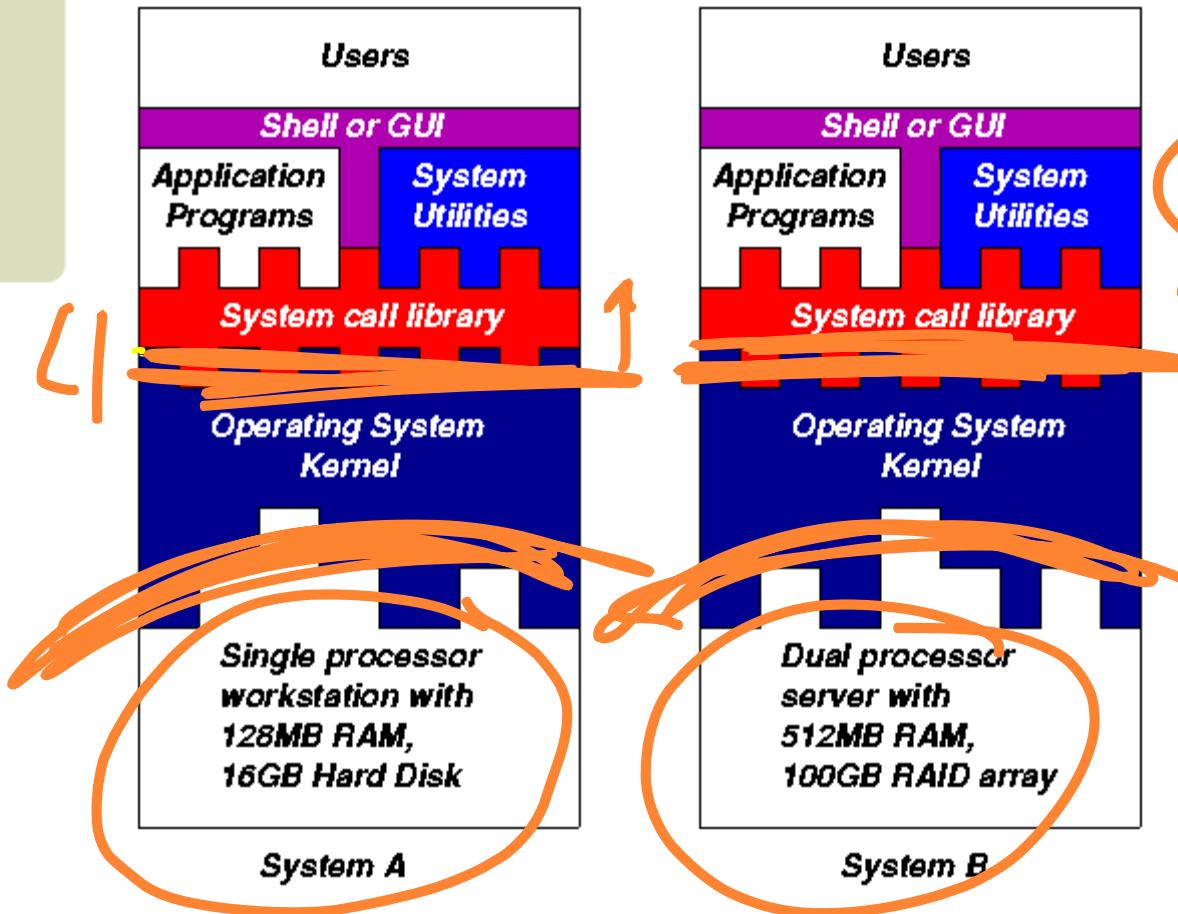


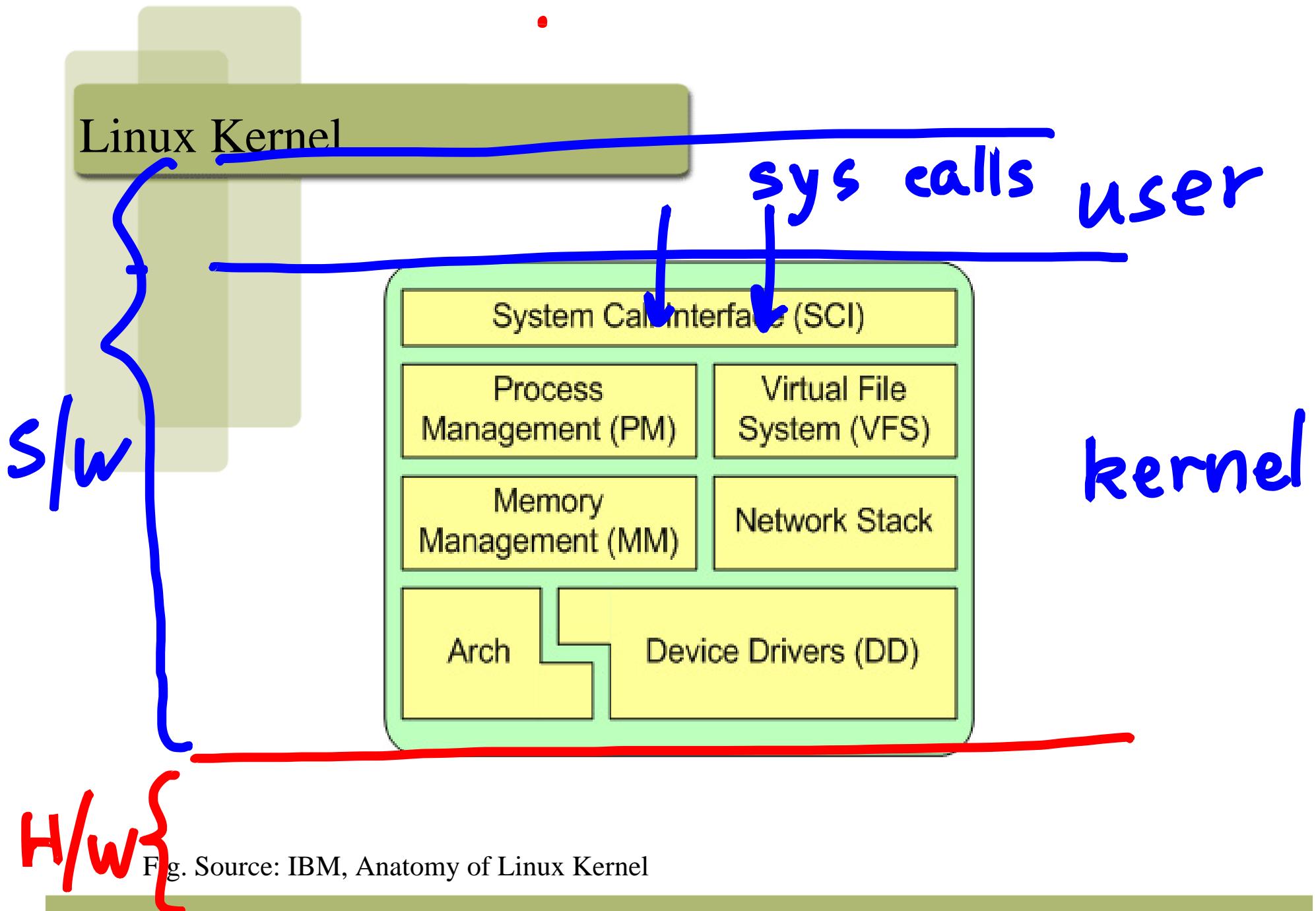
Fig. Source: IBM, Anatomy of Linux Kernel

## General view of O/S



✓ ported  
uniform  
interface

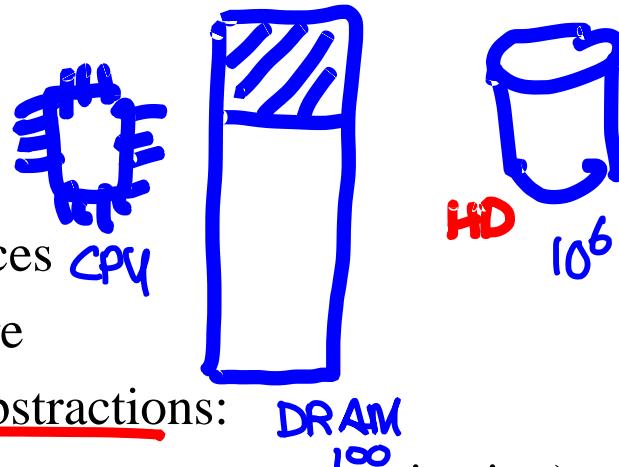
Fig. Source: W. Knottenbelt, UK



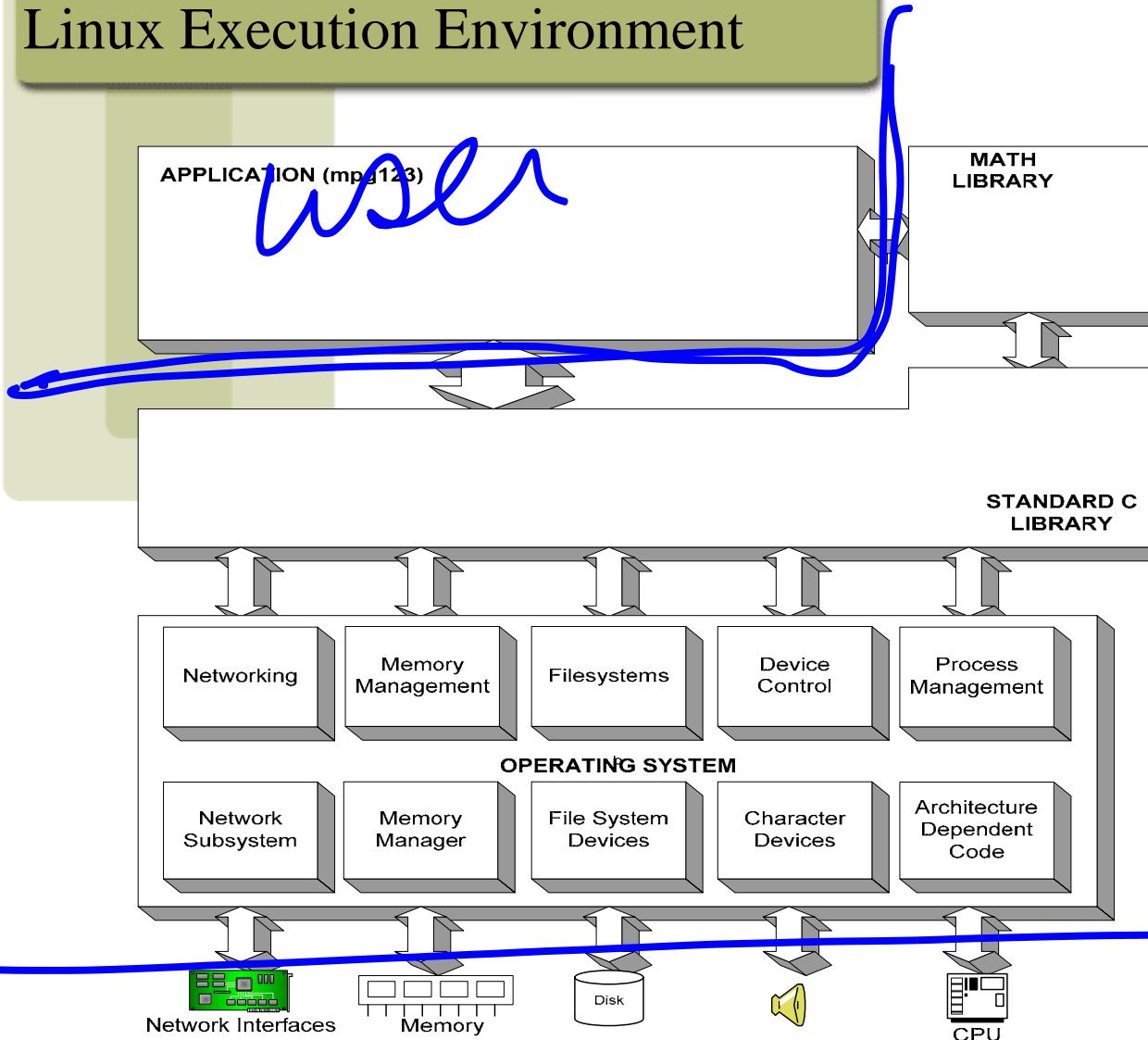
## The Operating System Kernel

- Resident in memory, privileged mode
- System calls offer general purpose services
- Controls and mediates access to hardware
- Implements and supports fundamental abstractions:
  - Process, file (file system, devices, interprocess communication)
- Schedules / allocates system resources: `malloc`
  - CPU, memory, disk, devices, etc.
- Enforces security and protection
- Event driven:
  - Responds to user requests for service (system calls)
  - Attends interrupts and exceptions
  - Context switch at quantum time expiration

resident in DRAM  
permanently

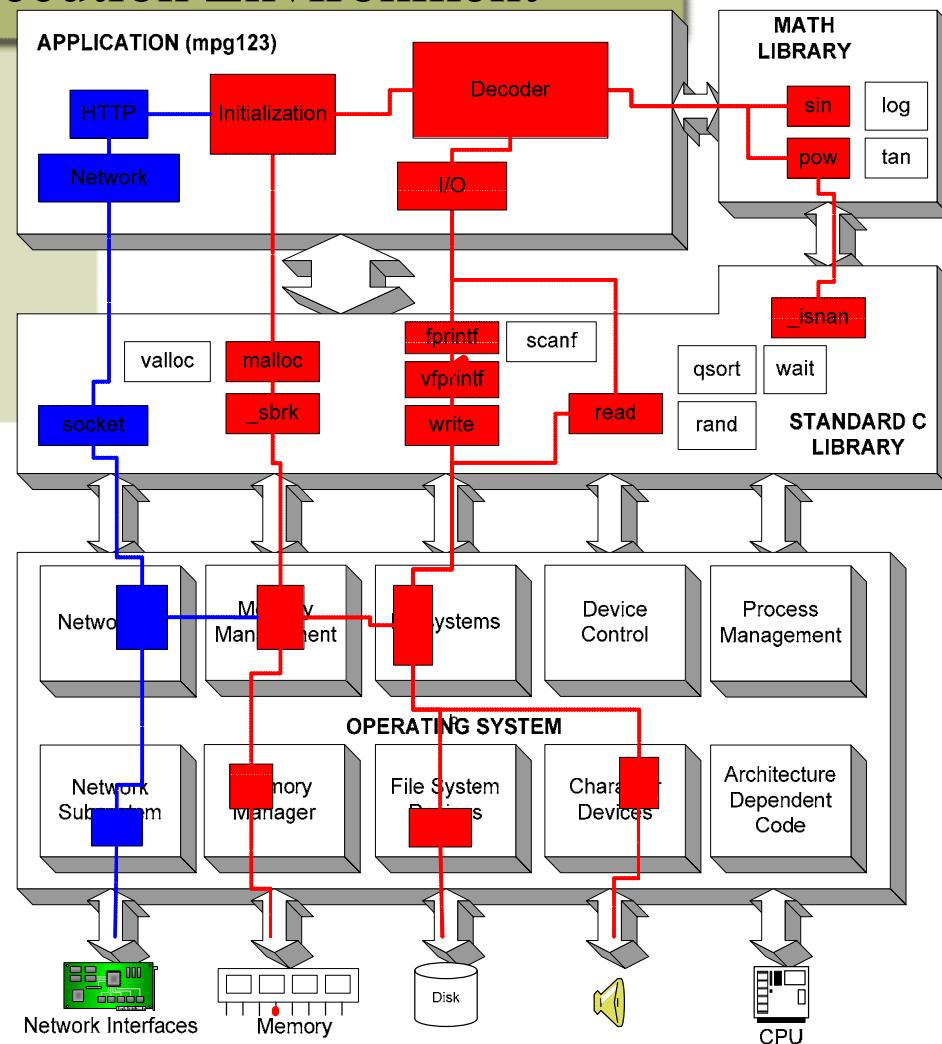


# Linux Execution Environment



- Program
- Libraries
- Kernel subsystems

# Linux Execution Environment

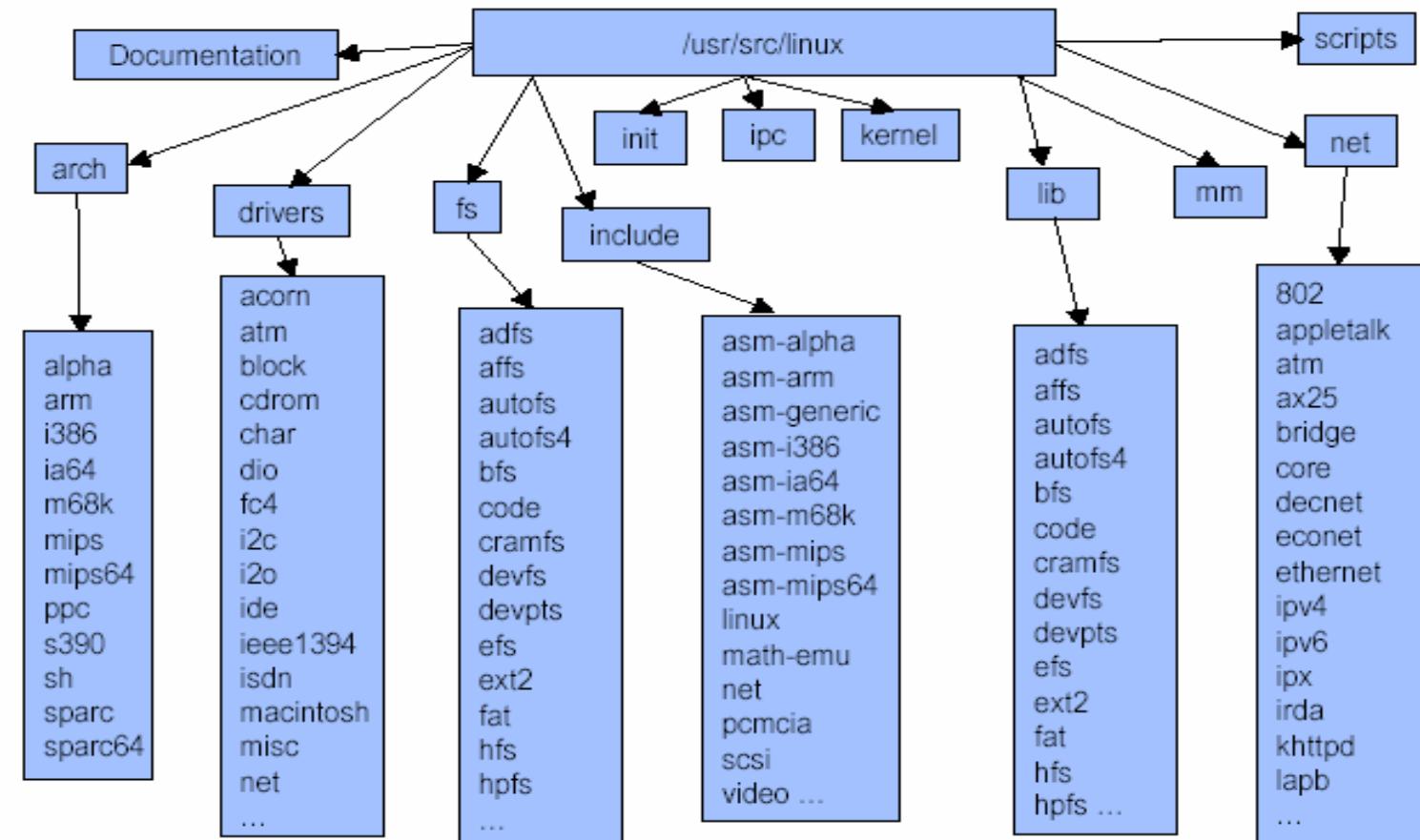


- Execution paths



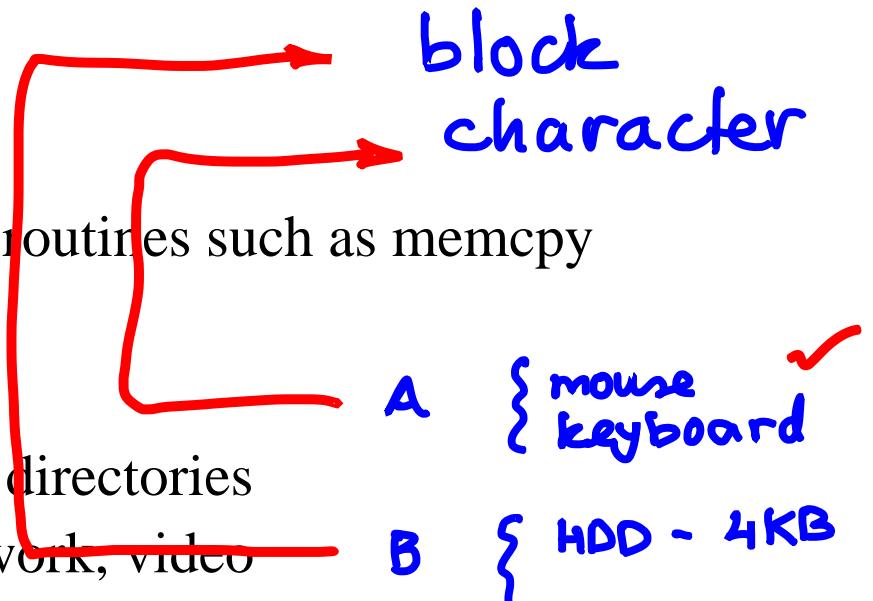
# Linux Source Layout

• C



# Linux Code Layout

- **Linux/arch**
  - Architecture dependent code.
  - Highly-optimized common utility routines such as memcpy
- **Linux/drivers**
  - Largest amount of code
  - Device, bus, platform and general directories
  - Character and block devices , network, video
  - Buses – pci, agp, usb, pcmcia, scsi, etc
- **Linux/fs**
  - Virtual file system (VFS) framework.
  - Actual file systems:
    - Disk format: ext2, ext3, fat, RAID, journaling, etc
    - But also in-memory file systems: RAM, Flash, ROM



# Linux Code Layout

- Linux/include

- Architecture-dependent include subdirectories.
- Need to be included to compile your driver code:
  - `gcc ... -I/<kernel-source-tree>/include ...`
- Kernel-only portions are guarded by #ifdefs

```
#ifdef __KERNEL__  
/* kernel stuff */  
#endif
```

- Specific directories: asm, math-emu, net, pcmcia, scsi, video.

*usr/src/linux*

## VFS and File Systems

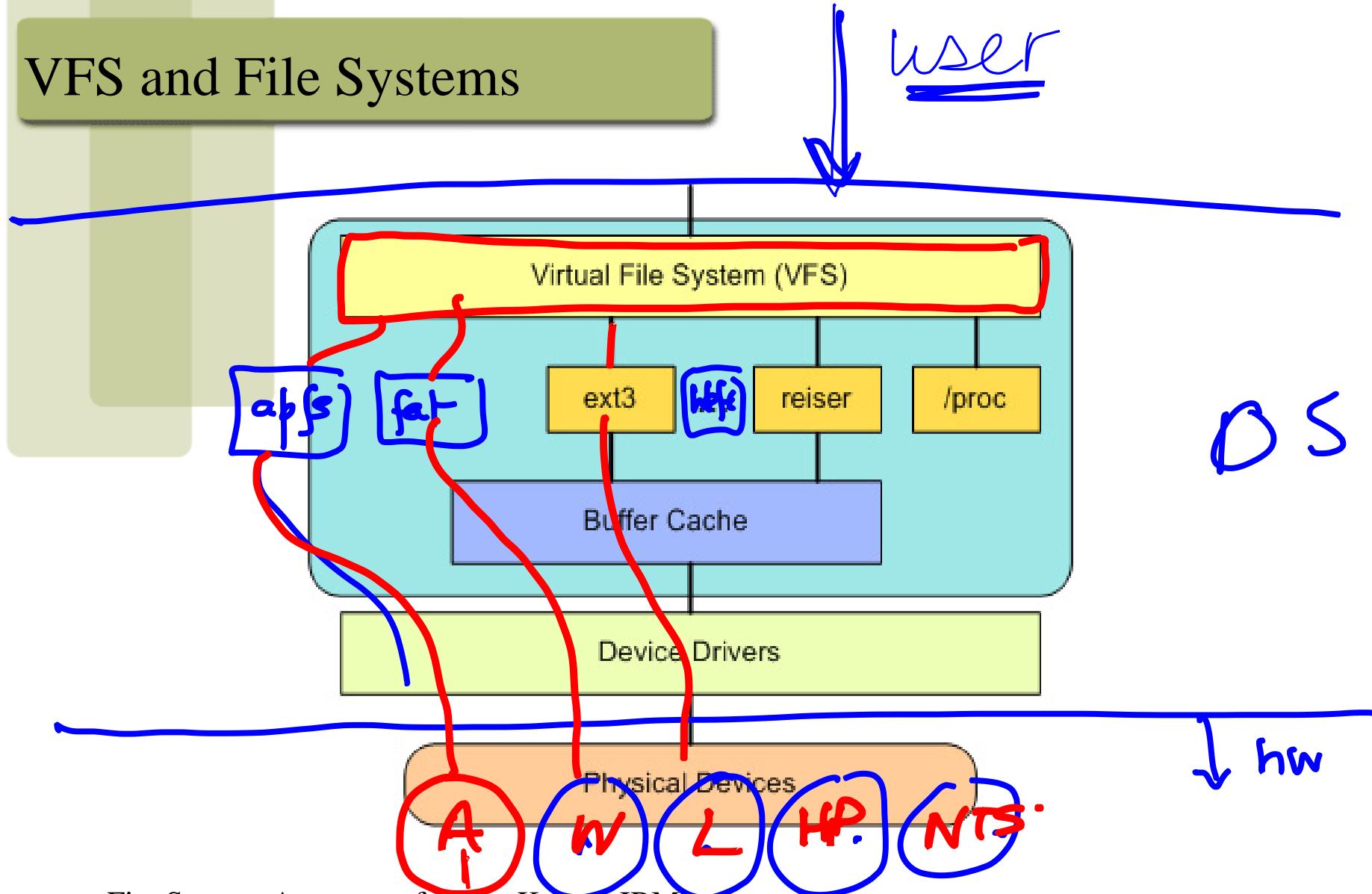
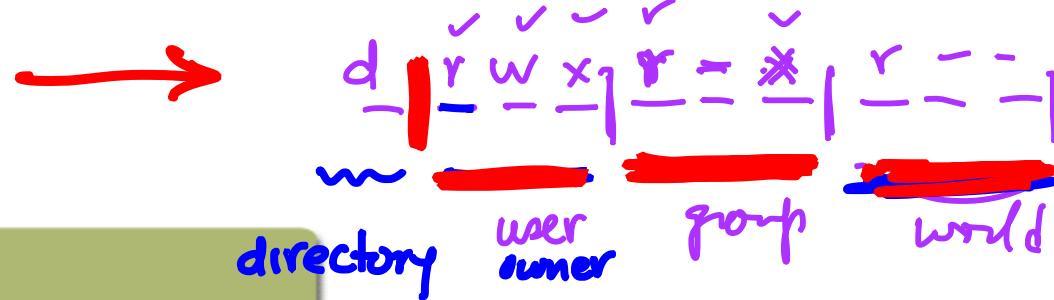


Fig. Source: Anatomy of Linux Kernel, IBM



## Linux commands

• ls: list directory contents *aka folder*

- ls : list directory contents
- ls -l : long or verbose option to ls  
 drwxr-xr-x 2 reddy faculty 4096 May 17 2005 bin/  
 d for directory, r for read, w for write x for executable  
 r-x : faculty can read and execute, but can't write  
 r-x: others can read and execute, but can't write  
 reddy is the owner, belongs to group "faculty"  
 size is 4096 bytes, created on May 17 2005, name of the directory  
 bin

\* -rw----- 1 reddy faculty 43231 Jun 20 2007 temp.txt

- mkdir: create a directory
- rm: remove a file
- rm -r : remove recursively
- rm -i: ask before removing or confirm
- vi or emacs : editors to open and edit files (**also vim**)
- cp : copy one file to another cp file1 file2: copies file1 to file2
- cat file1 : list the contents of file1 **to screen**
- more file1: list one screen of file1
- grep string file1 : list all the lines in file1 that contain string
- mv file1 file2 : move file1 to file2 i.e., rename file1 to file2, remove earlier file if file2 exists.

mkdir newdir

rm -r newdir

grep how hello.txt

1: hello how

3: do how

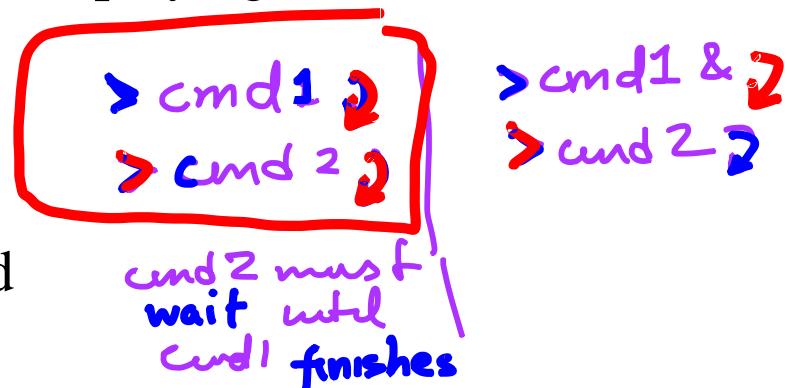
hello how  
do you  
do how

hello.txt

## Linux commands

- diff file1 file2 : find the differences between file1 and file2
- grep string file1 | wc
  - Look for string in file1 and do “word count” on those lines of the file
- Command1 | command2
  - Run command1 and “pipe” the result into command2
- grep string file1 > file2
  - Store the result of grep into file2 instead of displaying them on the terminal
- grep string file1 &
  - Run the command in the background
  - Can run other commands in the foreground
  - Or use the terminal for other things

cmd1 | cmd2  
"run cmd1, send output as input to cmd2"



## Reentrant code

- A function or code that can be reentered *safely*
  - It can be executed safely concurrently
- Int i;  
Function f()  
{  
    i = i+2;  
    return(i);  
}
- Returned value i can change if two threads enter f concurrently
  - Not reentrant

## Reentrant code

- function f(int i)  
{  
    int temp = i;  
    return (temp+2);  
}
- f() return value is only based on how it is called f(1), f(2) ...
  - Not on the order in which it is called or how many threads are executing f()

## Process and System Calls

- Process: program in execution. Unique “pid”. Hierarchy.
- User address space vs. kernel address space
- Application requests OS services through TRAP mechanism
  - x86: syscall number in eax register, exception (int \$0x80)
  - result = read (file descriptor, user buffer, amount in bytes)
  - Read returns real amount of bytes transferred or error code (<0)
- Kernel has access to kernel address space (code, data, and device ports and memory), and to user address space, but only to the process that is currently running
- “Current” process descriptor. “current → pid” points to current pid
- Two stacks per process: user stack and kernel stack
- Special instructions to copy parameters / results between user and kernel space

process id

## Exceptions and Interrupts

- Hardware switch to kernel mode. Uses “interrupt stack”.
- Synchronous exceptions: page fault, illegal instruction, etc
  - Triggered in the context of the current process
  - Can access user space; have a default kernel recovery action
  - Can be managed by the process itself (signal handling)
    - Signal (signum, handler) [ signal(SIGSEGV,invalid\_mem\_handler) ]
    - Can also be initiated by the kernel or other (related) process
      - Kill (pid, signum) [ kill(1234, SIGSTOP) ]
- Asynchronous interrupts: devices, clock, etc
  - Not in the context of the related process → no access to user memory, must buffer data in kernel space
  - Can signal a process (will be handled when scheduled to run)
- Traps, exceptions, interrupts can trigger process scheduling

## Scheduling and Exception Delivering

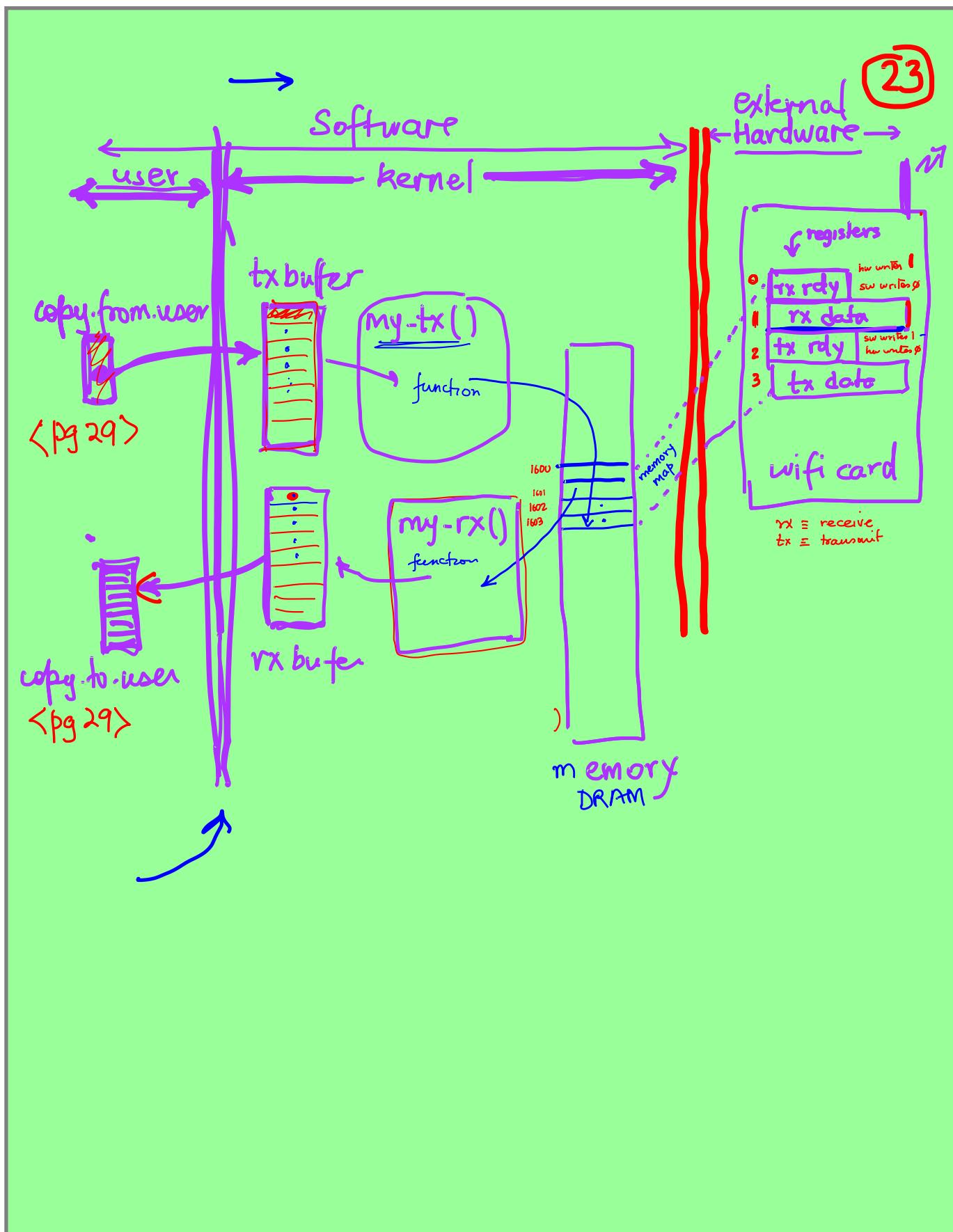


- Kernel is non **preemptible** (changed in Linux 2.6), but is multithreaded and multiprocessor: concurrency and parallelism
- Kernel state needs to be coherent before exit to user mode:
  - Process pending signals are checked and handlers are called
  - Context switch if current process is no longer at highest priority
  - Zombie (dead) process final dispositions, deallocation of resources and notification to related living ones
  - If no process to run, switch to kernel idle thread

nop

## Kernel Modules

- Kernel modules are inserted and unloaded dynamically
  - Kernel code extensibility at run time
  - `insmod` / `lsmod`/ `rmmod` commands. Look at `/proc/modules`
  - Kernel and servers can detect and install them automatically, for example, `cardmgr` (pc card services manager)
- Modules execute in kernel space
  - Access to kernel resources (memory, I/O ports) and global variables (look at `/proc/ksyms`)
  - Export their own visible variables, `register_symtab ()`;
  - Can implement new kernel services (new system calls, policies) or low level drivers (new devices, mechanisms)
  - Use internal kernel basic interface and can interact with other modules (pcmcia memory\_cs uses generic card services module)
  - Need to implement `init_module` and `cleanup_module` entry points, and specific subsystem functions (`open, read, write, close, ioctl ...`)  **fops**



## Hello World

- hello\_world\_module.c:

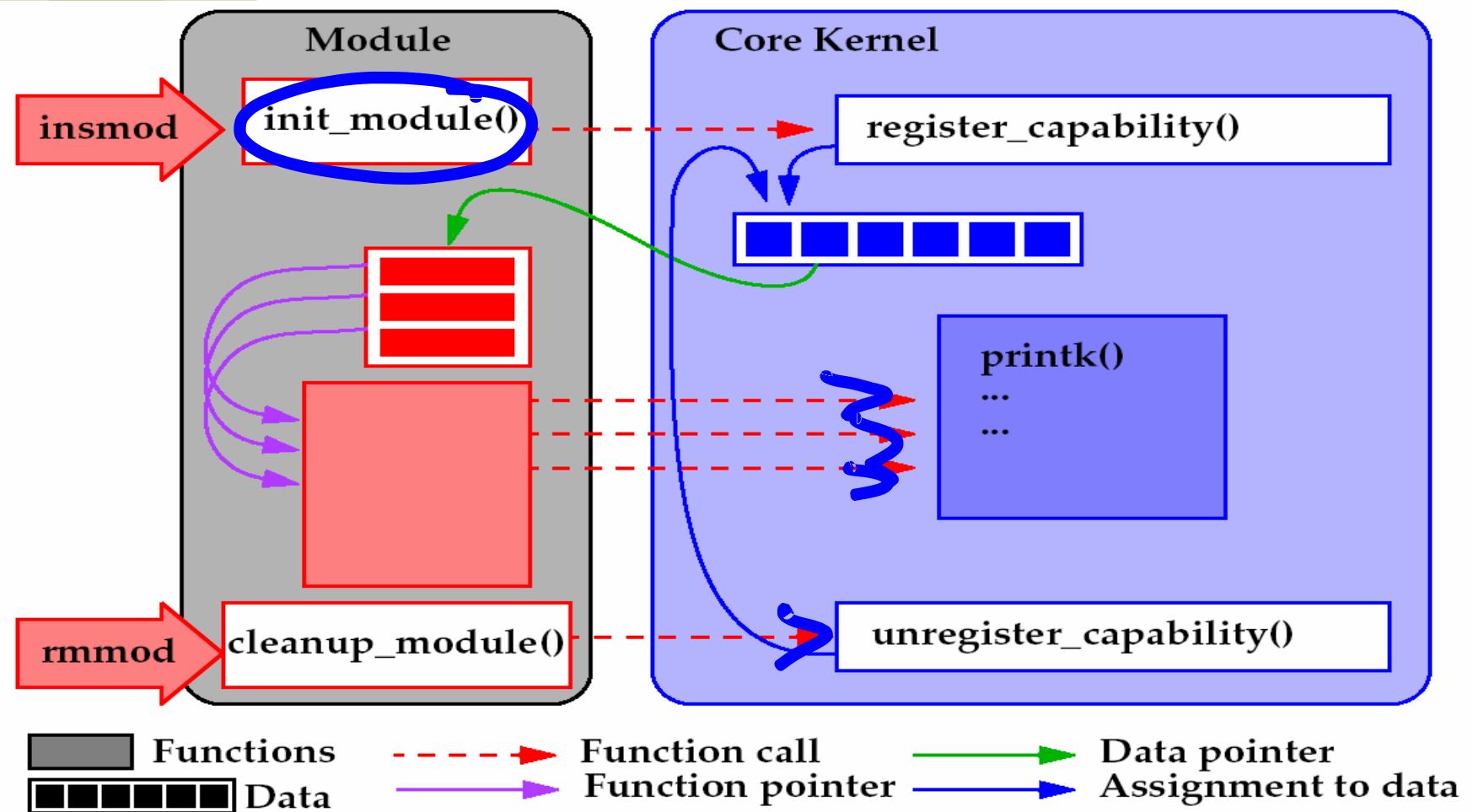
```
#define MODULE

#include <linux/module.h>
static int __init init_module(void)
{
    printk("<1>Hello, world\n"); /* <1> is message priority. */
    return 0;
}
static int __exit cleanup_module(void)
{
    printk("<1>Goodbye cruel world\n");
}
```

red

- **Printk** (basic kernel service) outputs messages to console and/or to /var/log/messages
- Use “**insmod hello\_world\_module.o**” to load it into the kernel space

## Linking a module to the kernel (from Rubini's book)



## Module programming

- Be careful: a kernel fault is fatal to the current process and sometimes the whole system
- Modules should support *concurrency* (support calls by different processes). Distinct data structures for each process (since the same code is executed) to ensure data is not corrupted.
- Driver code must be *reentrant*: keep status in local (stack allocated) variables or dynamic memory allocation: `kmalloc / kfree`
- This allows the process executing to suspend (e.g., wait for pcmcia card interrupt) and other processes to execute the same code.
- It is not a good idea to assume your code won't be interrupted.
- `Sleep_on(wait_queue)` or `interruptible_sleep_on(wait_queue)` to yield the cpu to another process
- `/proc/ioports` contains information about registered ports. `/proc/iomem` contains info about I/O memory

# Modules and Device Drivers

- Device Driver is a loadable module that manages data transfers between device and O/S.
  - Modules can be loaded and unloaded at boot time
  - A device driver can be used by other modules
  - Device driver must use standard entry points
  - Standard entry points are listed in struct file\_operations
  - Standard entry points struct file\_operations custom\_fops = {
    - llseek: NULL,
    - read: custom\_read,
    - write: custom\_write,
    - ioctl: NULL,
    - open: custom\_open,
    - release: custom\_release,
  - mmap: NULL
- };

## Different Operations

- Open: Opens and initializes the device internal structures
- my\_open(struct inode \*inode, struct file \*filp);
  - Register the device driver
  - Do initial setting up of driver structures
- my\_release(struct inode \*inode, struct file \*filp);
  - Release any resources
  - Unregister the device driver

## Different Operations

- `ssize_t read(struct file *filep, char *buf, size_t count, loff_t * offp);`
- `ssize_t write(struct file *filep, char *buf, size_t count, loff_t * offp)`
- `buf` is a buffer in user space
- When you want to transfer data from kernel to user space, use  
~~`copy_to_user (buf, kbuf, size)` copy data in kernel buffer `kbuf` to  
user buffer `buf` of size bytes~~
- Similarly, use ~~`copy_from_user(buf, kbuf, size)`~~ when you want to  
transfer data from user space to kernel space

# Device Drivers

- **Character Drivers:** deal with reading and writing one character at a time
  - Keyboard, line printer etc.
  - Not limited to “character” at a time
  - More flexible (see below)
- **Block Drivers:** deal with reading and writing blocks of data at a time
  - File systems, disk drives
  - All I/O done through buffer cache in kernel
- Network device drivers: deal with interacting with network interfaces

# Device Drivers

- Have Major number and Minor numbers
  - Major number could identify a type of device
  - Could be serviced by a single device driver
  - Minor number identifies a number of such devices
  - Together they identify a specific device

```
- crw-rw-rw- 1 root root 1, 3 Feb 23 1999 null
- crw----- 1 root root 10, 1 Feb 23 1999 psaux
- crw----- 1 rubini tty4, 1 Aug 16 22:22 tty1
- crw-rw-rw- 1 root dialout 4, 64 Jun 30 11:19 ttyS0
- crw-rw-rw- 1 root dialout 4, 65 Aug 16 00:00 ttyS1
- crw----- 1 root sys 7, 1 Feb 23 1999 vcs1
- crw----- 1 root sys 7, 129 Feb 23 1999 vcsa1
- crw-rw-rw- 1 root root 1, 5 Feb 23 1999 zero
```

... 210 20 ... wifi

/dev contents

mouse

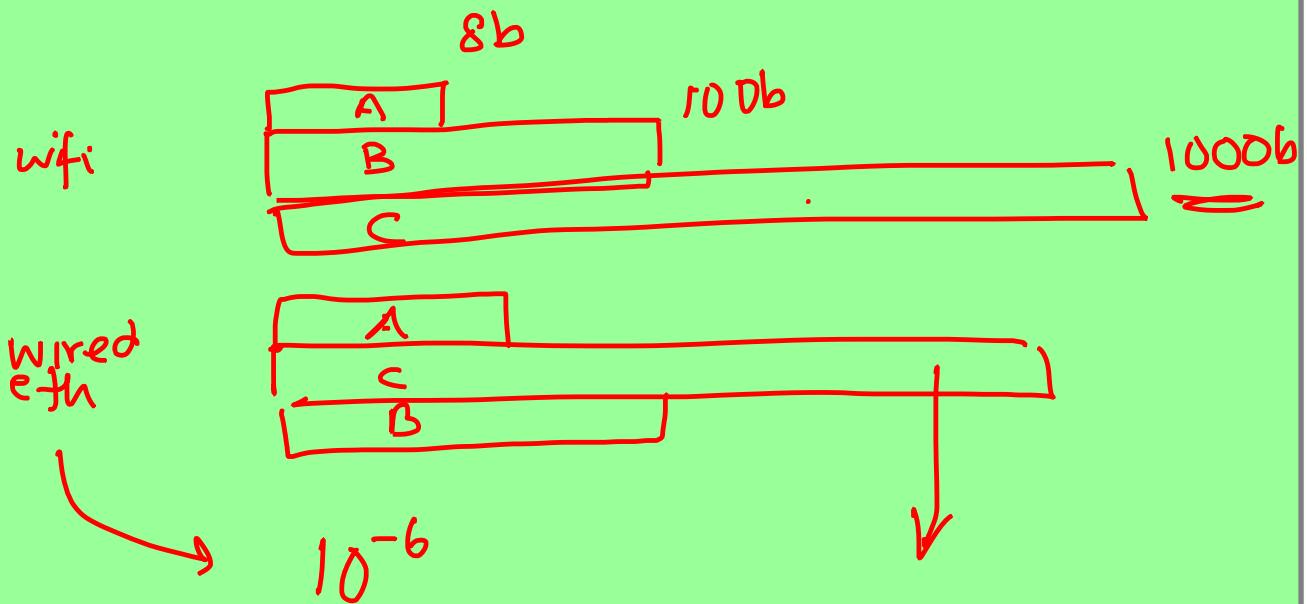
}

terminals

(30)

## Block/char devices

$10^{-3}$  = BER  
bit error rate



## elegant abstractions

(31)

- ① device registers mapped to mem
- ② devices mapped to filenames.

/home/sunil/hw3.pdf } rep file

/dev/mouse

/dev/wifi

device

user just does "read" or  
"write" to this "file"



register  
w/ fops <slide 27>  
of the  
driver

pg 32 (xx)  
mknod /dev/wifi  
c 210 20

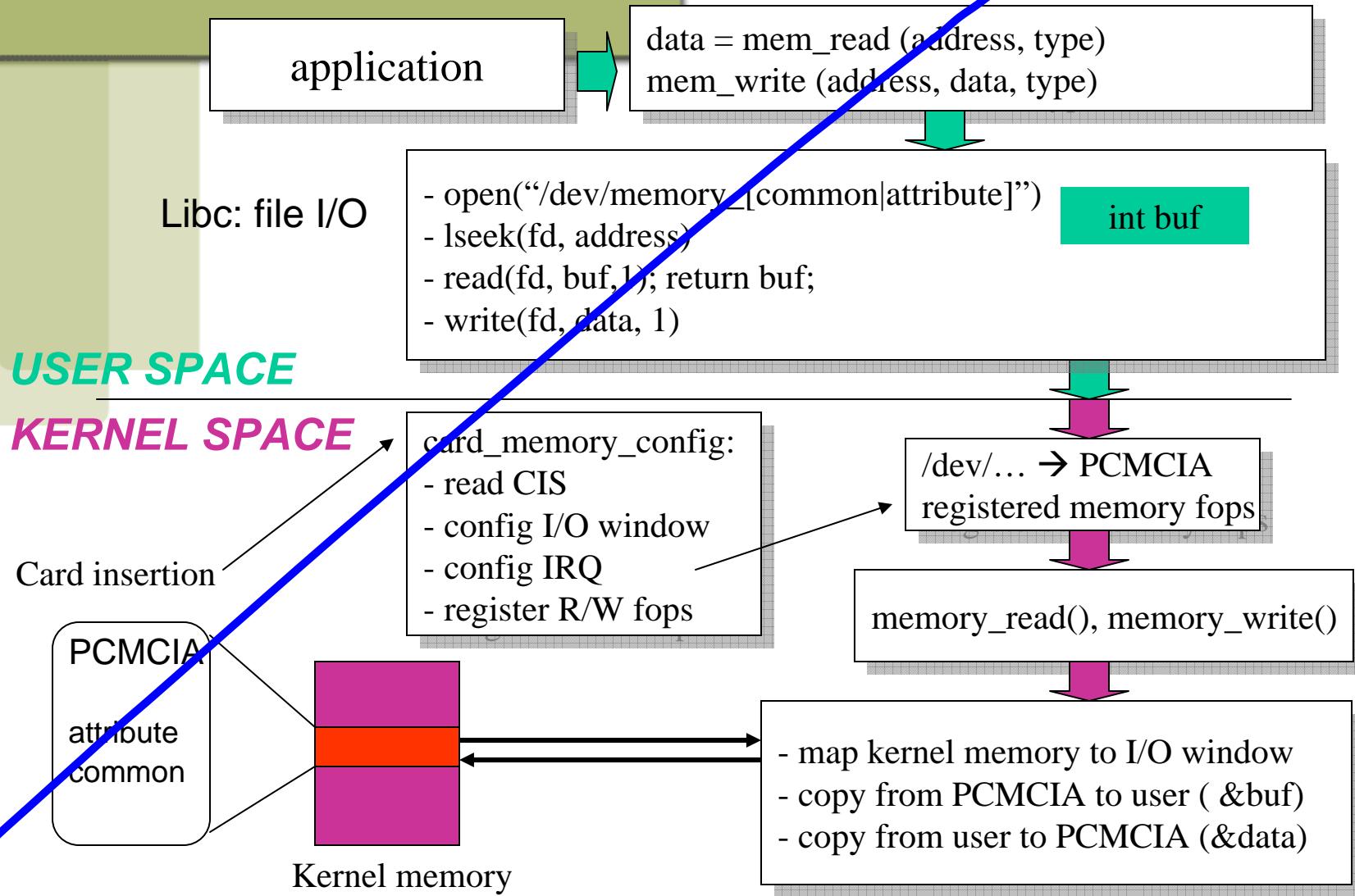
so if I do a "read" on  
this device, it would run the  
"custom read" of the fops

## Register Capability

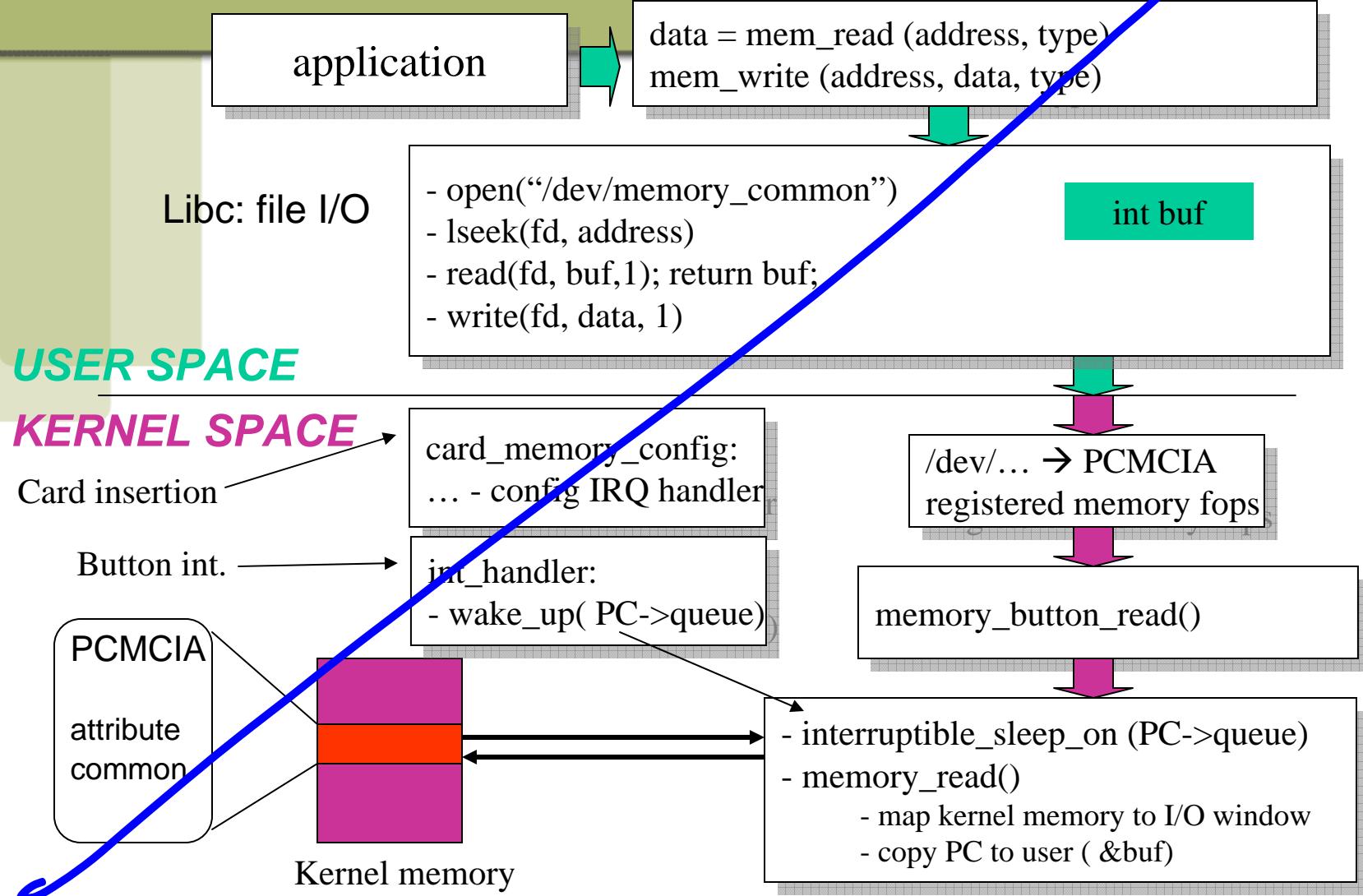
- You can register a new device driver with the kernel:
  - `int register_chrdev(unsigned int major, const char *name, struct file_operations *fops);`
  - A negative return value indicates an error, 0 or positive indicates success.
  - *major*: the major number being requested (a number < 128 or 256).
  - *name*: the name of the device (which appears in `/proc/devices`).
  - *fops*: a pointer to a global jump table used to invoke driver functions.
- Then give to the programs a name by which they can request the driver through a device node in `/dev`
  - To create a char device node with major 254 and minor 0, use:
    - `mknod /dev/memory_common c 254 0`
  - Minor numbers should be in the range of 0 to 255.



## PCMCIA Read/Write Common/Attribute Memory



## PCMCIA “Button Read” Interrupt handling



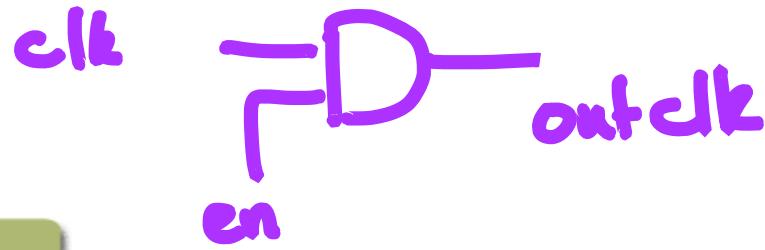
## Links for Driver Developers

- “Linux Device Drivers”, Alessandro Rubini & Jonathan Corbet, O’Reilly, 2nd Edition
  - On-line version: <http://www.xml.com/ldd/chapter/book>
  - Book examples: <http://examples.oreilly.com/linuxdrive>
- PCMCIA programming:
  - <http://pcmcia-cs.sourceforge.net/ftp/doc/PCMCIA-HOWTO.html>
  - <http://pcmcia-cs.sourceforge.net/ftp/doc/PCMCIA-PROG.html>
- Linux :
  - Kernel archives: <http://www.kernel.org>
  - Linux for PDA: <http://handhelds.org>
  - Cross-Referencing: <http://lxr.linux.no/source>



## Design Pitfalls

- Making things too complex
  - I've said it before, I'll probably say it again. This is really critical
- Not using one clock signal everywhere
  - This is really important.
    - Multiple clocks can create Heisenbugs that only appear some of the time
    - Worse yet, can create bugs that only appear in hardware vs. being testable in simulation



## Common Logic Problems

- INOUT Ports
  - If you have a port of type INOUT in a module, you need to explicitly drive it to high impedance “z” when the module is not driving the port.
    - Simply failing to specify an output value for the port isn’t good enough, you can get flaky signal or even fire hazard
- Gated clocks
  - Your design should have one clock. It should go to all clock pins on all modules.
  - Never, ever, put any logic between the clock and the clock pin of a module
  - Never, ever, connect anything but the global clock to the clock pin of a module.