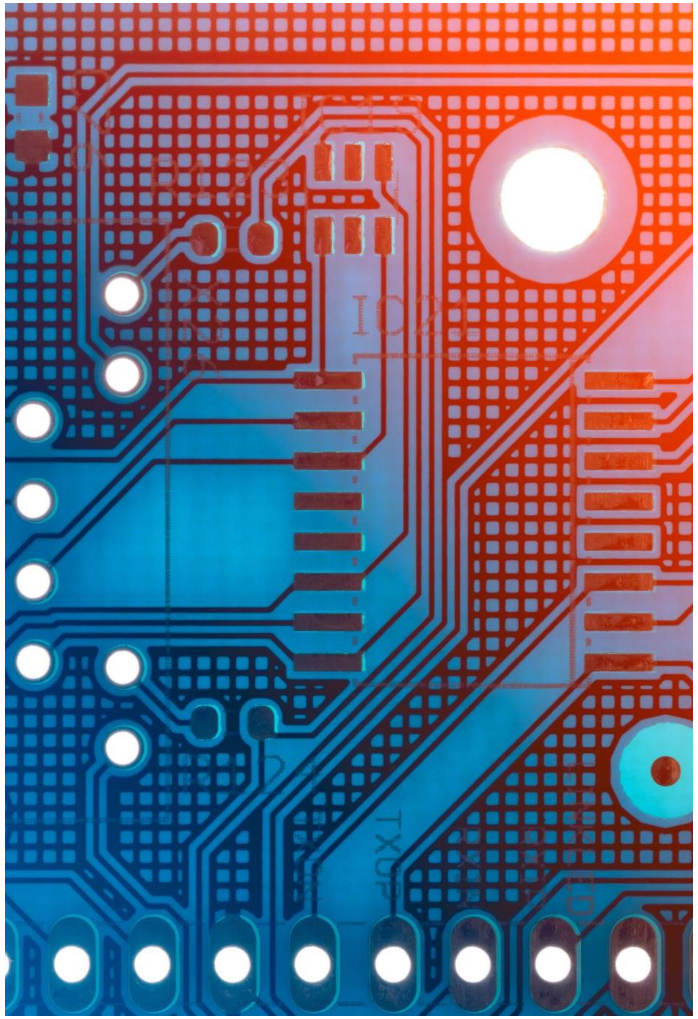
Linux Device Drivers

Dr. Vimal Baghel
Assistant Professor
SCSET, BU

Outline

- System Call Review
- Introduction to Device Drivers
- Why Device Drivers?
- Kernel & Device Drivers
- Policy Independence
- Kernel Functions
- Classes of Devices and Modules
- Security Issues
- Version Numbering & License Issues
- Q&A

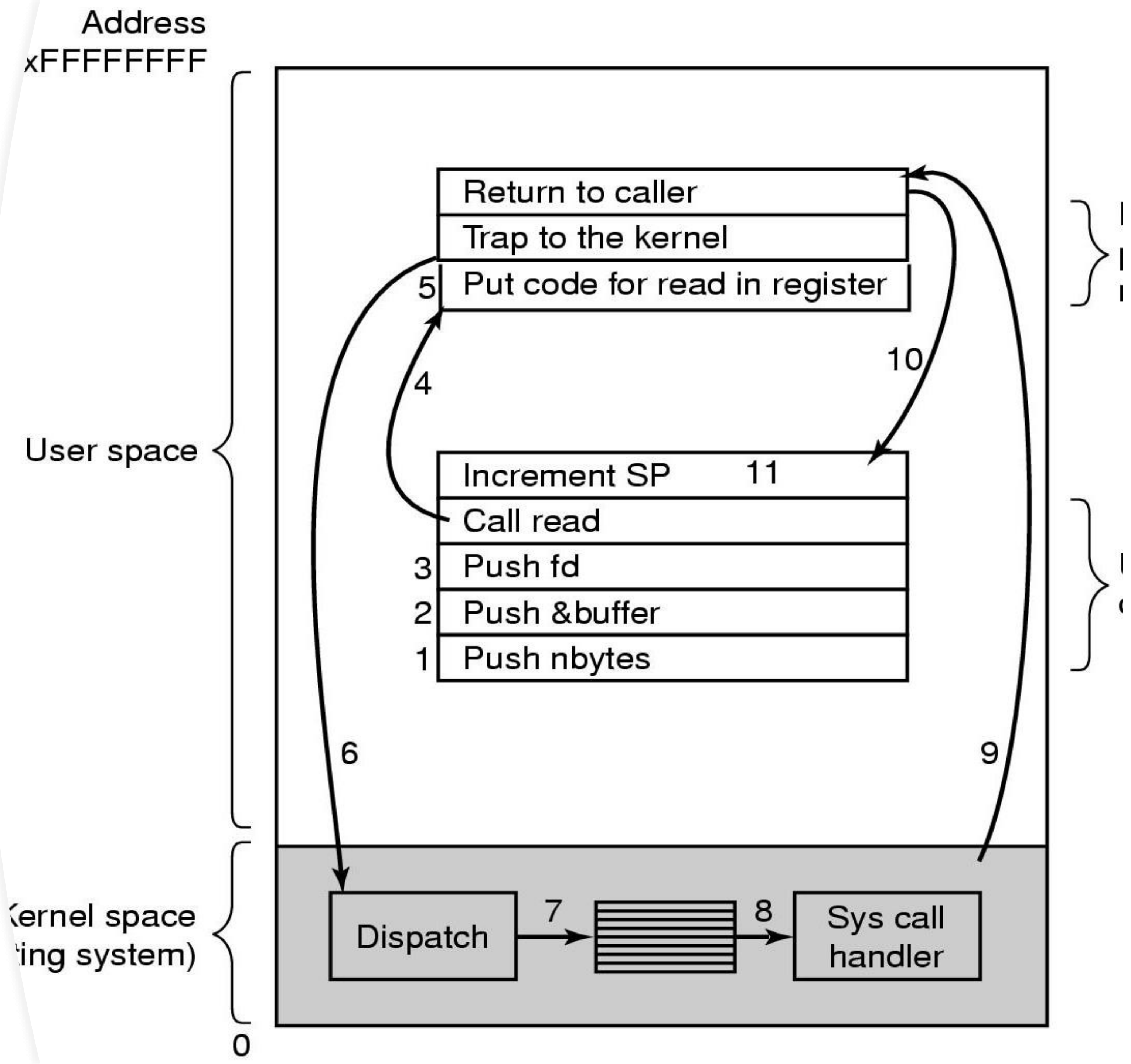


System Calls

The mechanism used by an application program to request service from the operating system.

System calls often use a special machine code instruction which causes the processor to change mode (e.g. to "supervisor mode" or "protected mode").

This allows the OS to perform restricted actions such as accessing hardware devices or the memory management unit.



Steps in Making a System Call

- ✓ There are 11 steps in making the system call
- ✓ read (fd, buffer, nbytes)

Some System Calls For Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Some System Calls For File Management

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Some System Calls For Directory Management

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Some System Calls For Miscellaneous Tasks

Miscellaneous

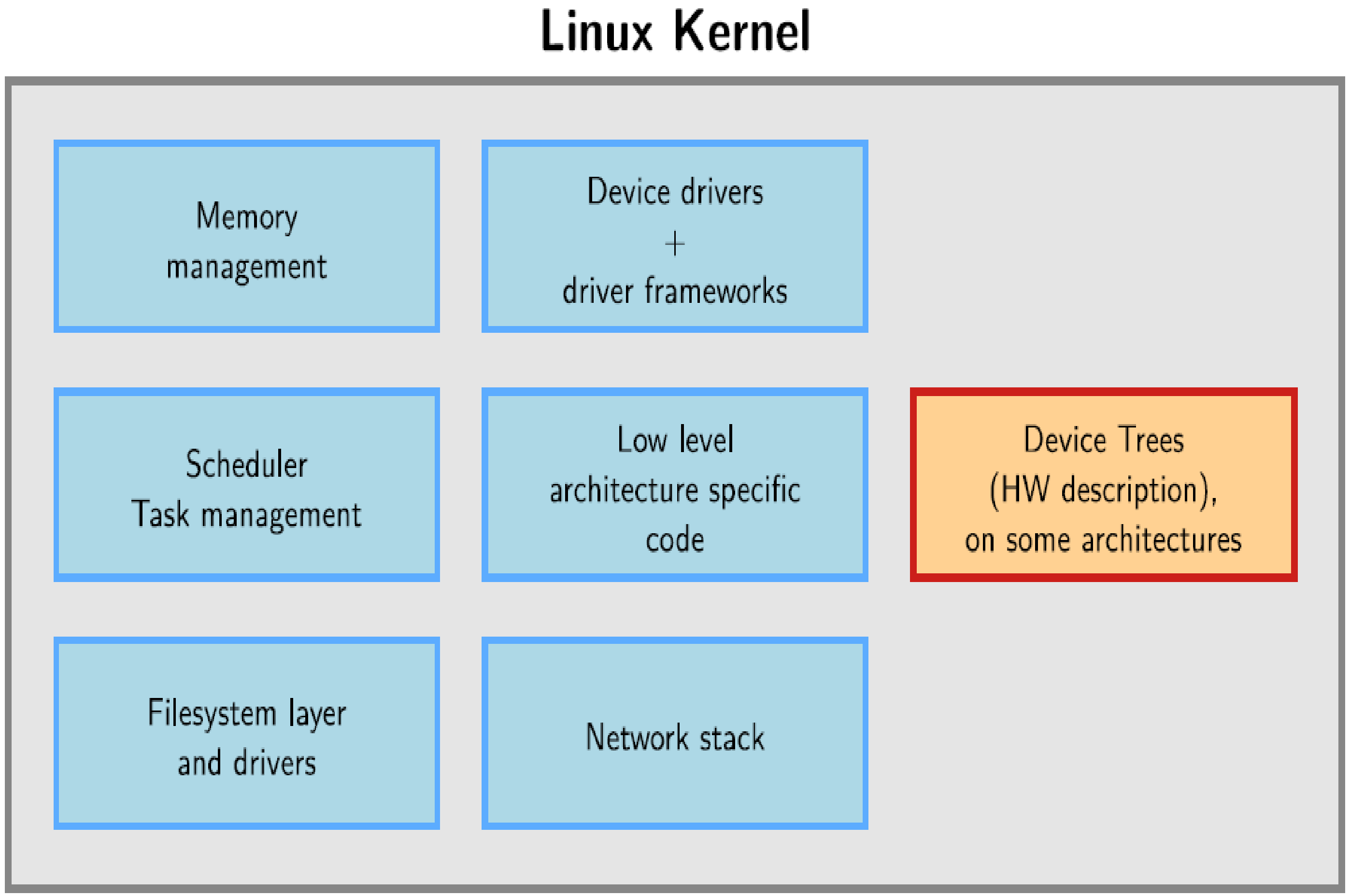
Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Introduction to Linux Device Drivers

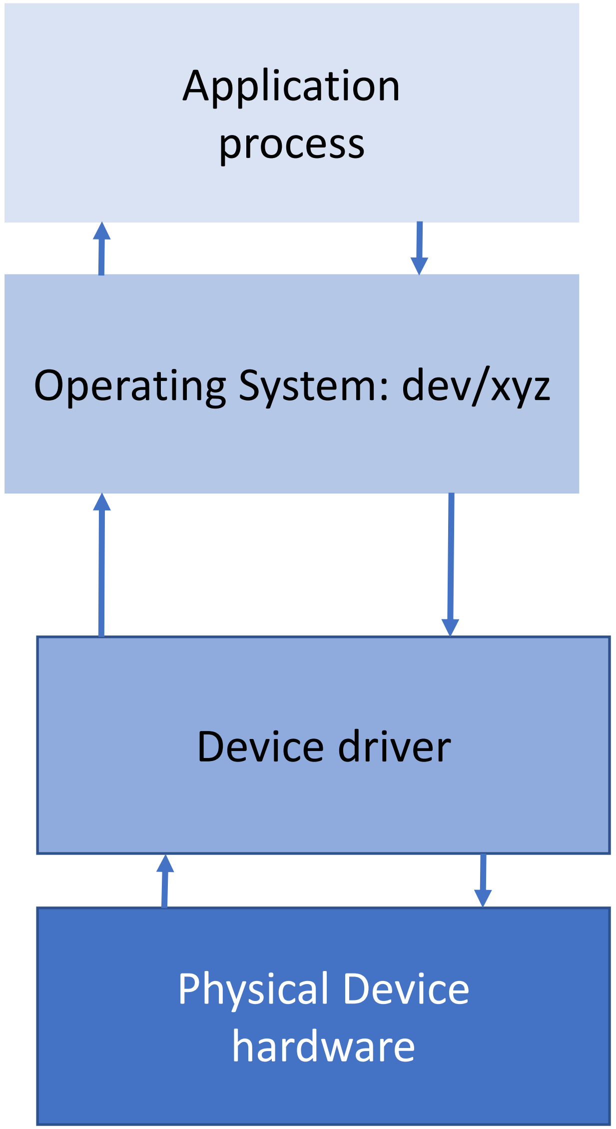
✓Device drivers are the “black boxes” that make a particular piece of hardware respond to a well-defined internal programming interface

✓they hide completely the details of how the device works

- Device drivers
  - Software interface to hardware device
  - Use standardized calls
    - Independent of the specific driver
  - Main role
    - Map standard calls to device-specific operations
  - Can be developed separately from the rest of the kernel
    - Plugged in at runtime when needed



Why Device Drivers?

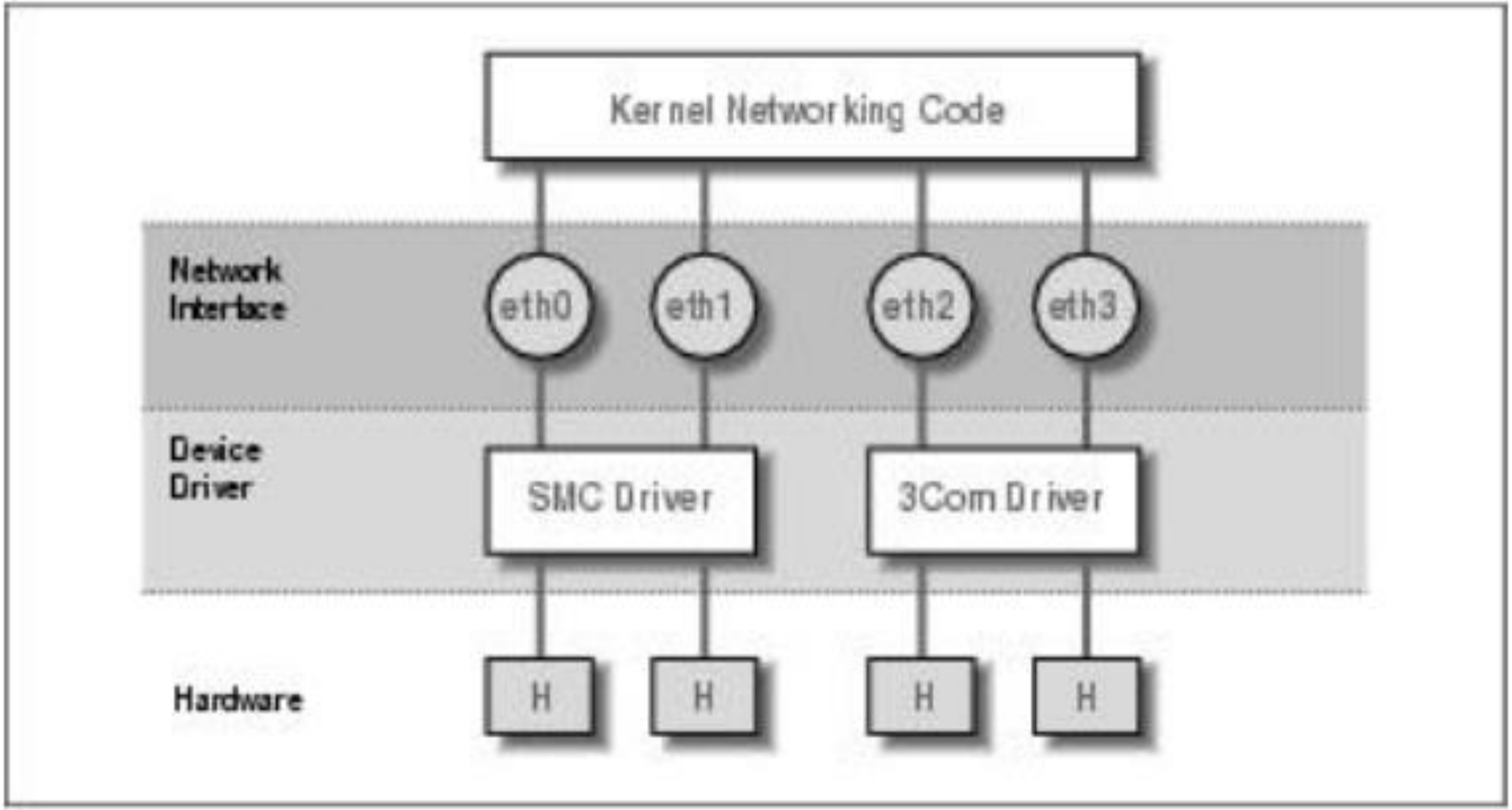


- Drivers help the hardware devices interact with the Linux Kernel.

Role of the Device Driver

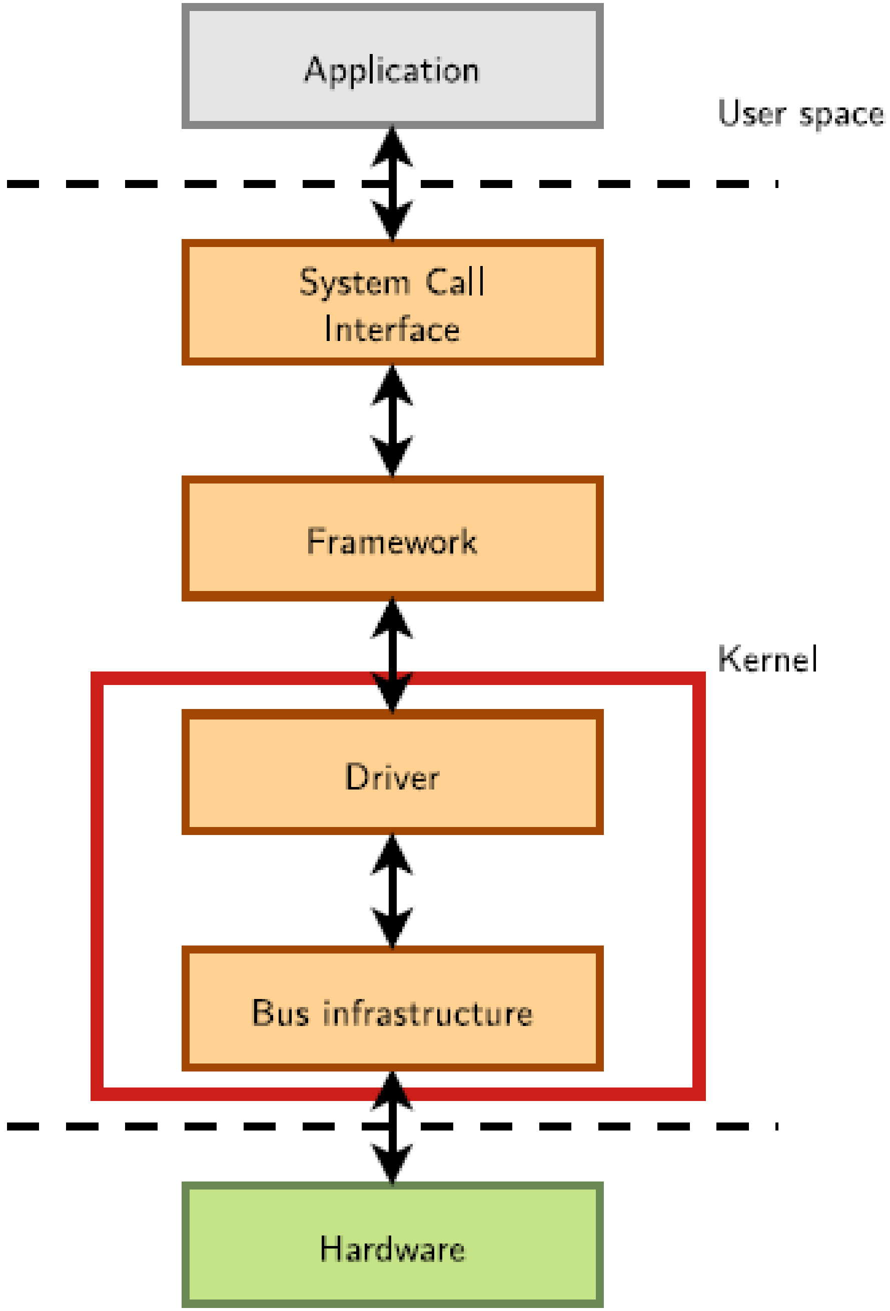
- Implements the *mechanisms* to access the hardware
  - E.g., Hard disk
    - Read/write blocks of data
    - Access as a continuous array
- Does not force particular *policies* on the user
  - Examples
    - Who and How many access the drive
    - Whether the drive is accessed via a file system
    - Whether users may mount file systems on the drive

The relationship between drivers, interfaces, and hardware



- When booting, the kernel displays the devices it detects and the interfaces it installs.

Kernel & Device Drivers



- In Linux, a driver is always interfacing with:
  - ▶ a **framework** that allows the driver to expose the hardware features in a generic way.
  - ▶ a **bus infrastructure**, part of the device model, to detect/communicate with the hardware.

Policy-Free Implementation

- Simplifies the design
- Separation of concerns
  - Capabilities provided
  - Use of Capabilities
- Reuse
  - Different policies do not require changes to the kernel

Splitting the Kernel

Kernel handles resource requests

Process management

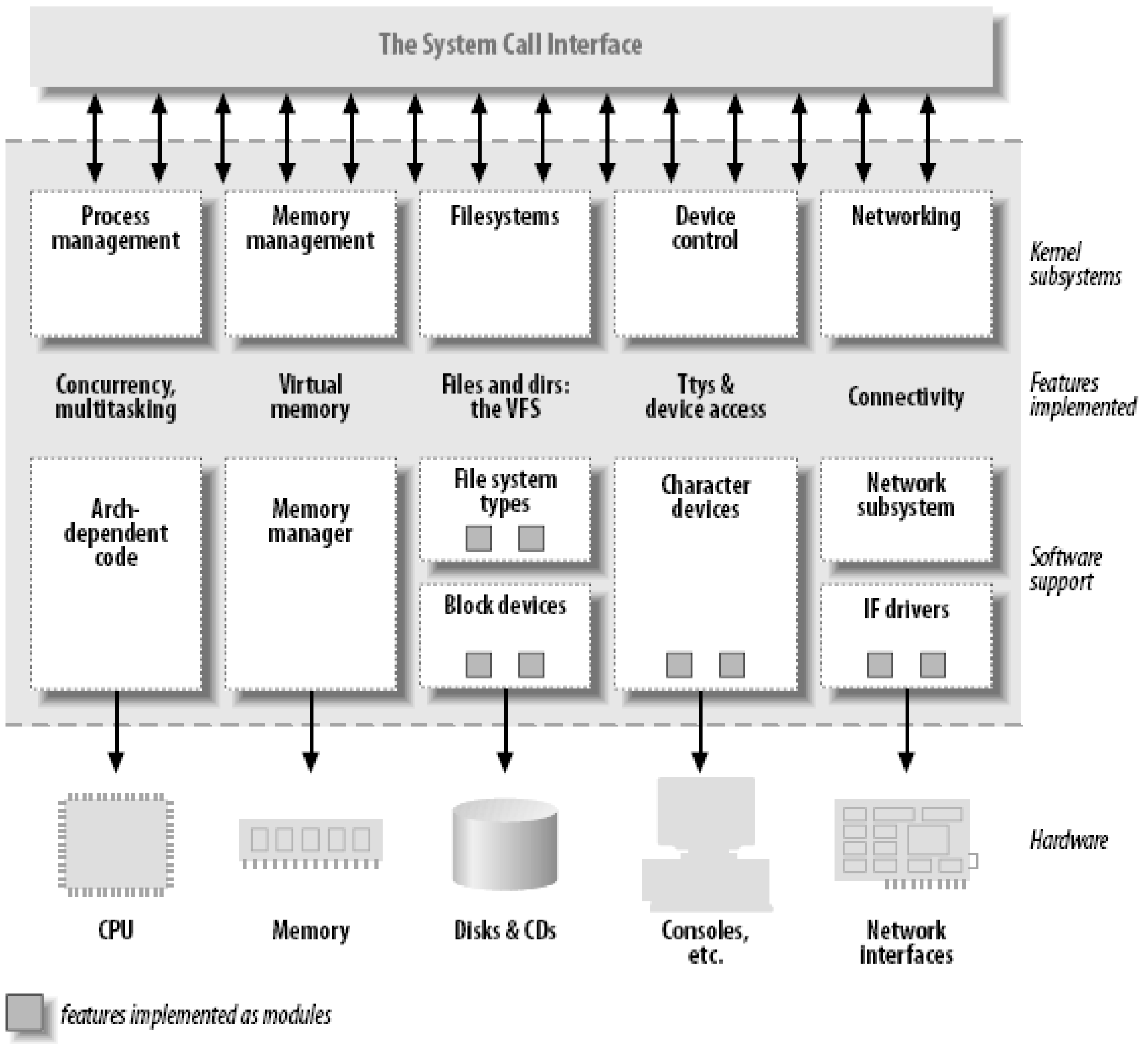
- Creates, destroys processes
- Supports communication among processes
- Signals, pipes, etc.
- Schedules how processes share the CPU/cores

Memory management

- Virtual addressing

Splitting the Kernel

- File systems
  - Everything in UNIX can be treated as a file
  - Linux supports multiple file systems
- Device control
  - System operation maps to a physical device
- Networking
  - Handles packets
  - Routing and network address resolution



Hardware Management using Device Drivers

- Any device that the Linux system must communicate with ***needs driver code inserted inside the kernel code.***
- The driver code allows the kernel ***to pass data back and forth to the device,*** acting as a middleman between applications and the hardware.
- Two methods are used for inserting device driver code in the Linux kernel:
  - Drivers compiled in the kernel
  - Driver modules added to the kernel

Hardware Management using Device Drivers

- Previously,
  - The only way to insert device driver code was to recompile the kernel.
  - Each time we added a new device to the system, We had to recompile the kernel code.
  - This process became even more inefficient as Linux kernels supported more hardware. Fortunately,
- Now,
  - Linux developers devised a better method to insert driver code into the running kernel.
  - Programmers developed ***the concept of kernel modules*** to allow you to insert driver code into a running kernel without having to recompile the kernel.
  - Also, a kernel module could be removed from the kernel when the device was finished being used.
  - This greatly simplified and expanded using hardware with Linux.

Runtime Loadable Kernel Modules

- The ability to add and remove kernel features at runtime
- Each unit of extension is called a *module*
- Use **insmod and modprobe** program to add a kernel module
- Use **rmmod** program to remove a kernel module

Classes of Devices and Kernel Modules

Character devices

Block devices

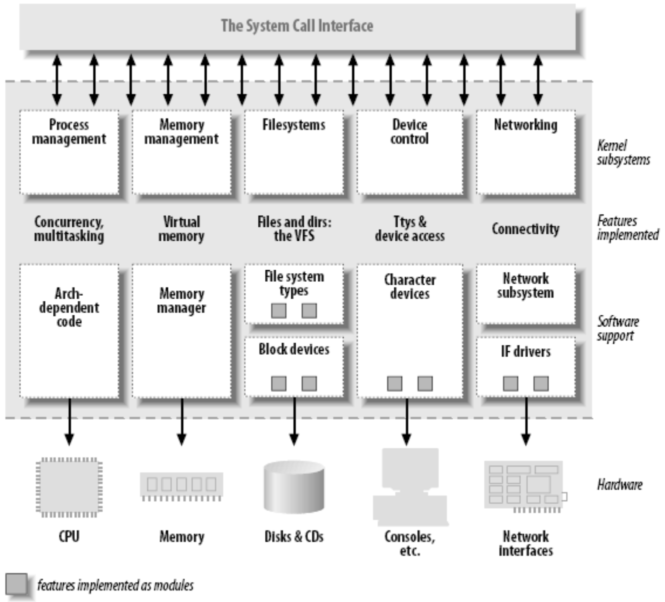
Network devices

Others

Character Module

Block Module

Network Module



Character Devices

Abstraction: a stream of bytes

Examples

- Text console (**/dev/console**)
- Serial ports (**/dev/ttyS0**)

Usually supports **open**, **close**, **read**, **write system calls**

Accessed sequentially (in most cases)

Might not support file seeks

Accessed through a filesystem node

Block Devices

Abstraction: continuous array of storage blocks (e.g. sectors)

Applications can access a block device in bytes

Accessed through a file system node

A block device can host a file system.

A block device can only handle I/O operations that transfer one or more whole blocks, which are usually 512 bytes (or a larger power of two) bytes in length.

Network Devices

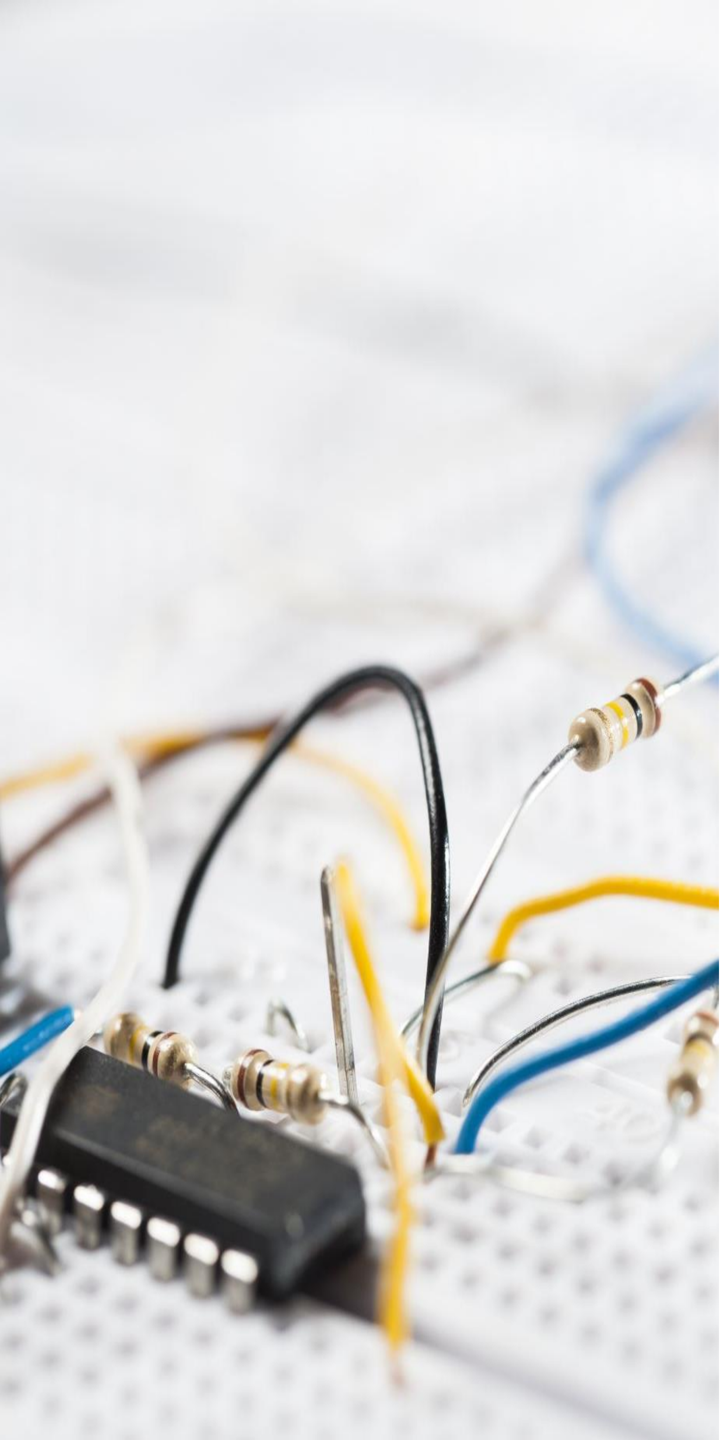
Abstraction: data packets

Send and receive packets

- Do not know about individual connections

Have unique names (e.g., **eth0**)

- Not in the file system
- Support protocols and streams related to packet transmission (i.e., no **read** and **write**)



Other Classes of Devices

- Examples that do not fit to previous categories:
  - USB
  - SCSI (Small Comp. System Interface)
  - FireWire
  - MTD (Memory Tech. Device)

Security Issues

- Deliberate vs. incidental damage
- Kernel modules present possibilities for both
- System does only rudimentary checks at module load time
- Relies on limiting privilege to load modules
  - And trusts the driver writers
- Driver writer must be on guard for security problems

Security Issues

- Do not define security policies
  - Provide mechanisms to enforce policies
- Be aware of operations that affect global resources
  - Setting up an interrupt line
    - Could cause another device to malfunction
  - Setting up a default block size
    - Could affect other users

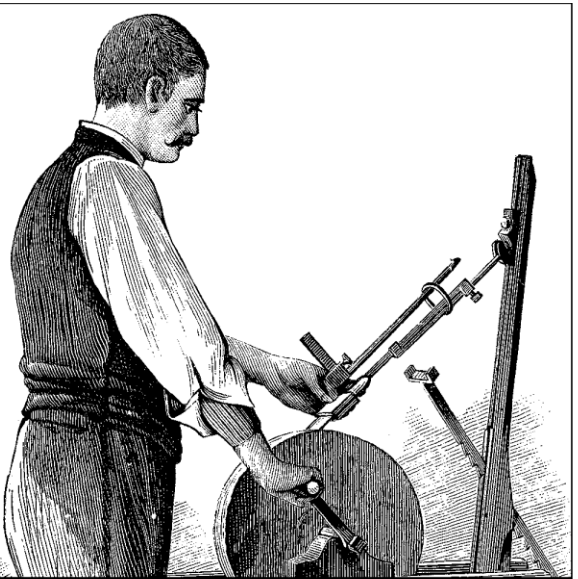
Security Issues

- Buffer overrun
  - Overwriting unrelated data
- Treat input/parameters with utmost suspicion
- Uninitialized memory
  - Kernel memory should be zeroed before being made available to a user
  - Otherwise, information leakage could result
    - Passwords
- Avoid running kernels/device drivers compiled by an untrusted friend
  - Modified kernel could allow anyone to load a module

Building and Running Kernel Modules

- Setting Up Your Test System

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void)
{
printk(KERN_ALERT "Hello, world\n");
return 0;
}
static void hello_exit(void)
{
printk(KERN_ALERT "Goodbye, cruel world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```



✓ This module defines two functions, one to be invoked when the module is loaded into the kernel (*hello_init*) and one for when the module is removed (*hello_exit*).

Thanks

Q & A