Drivers and input/output systems

Operating Systems Design

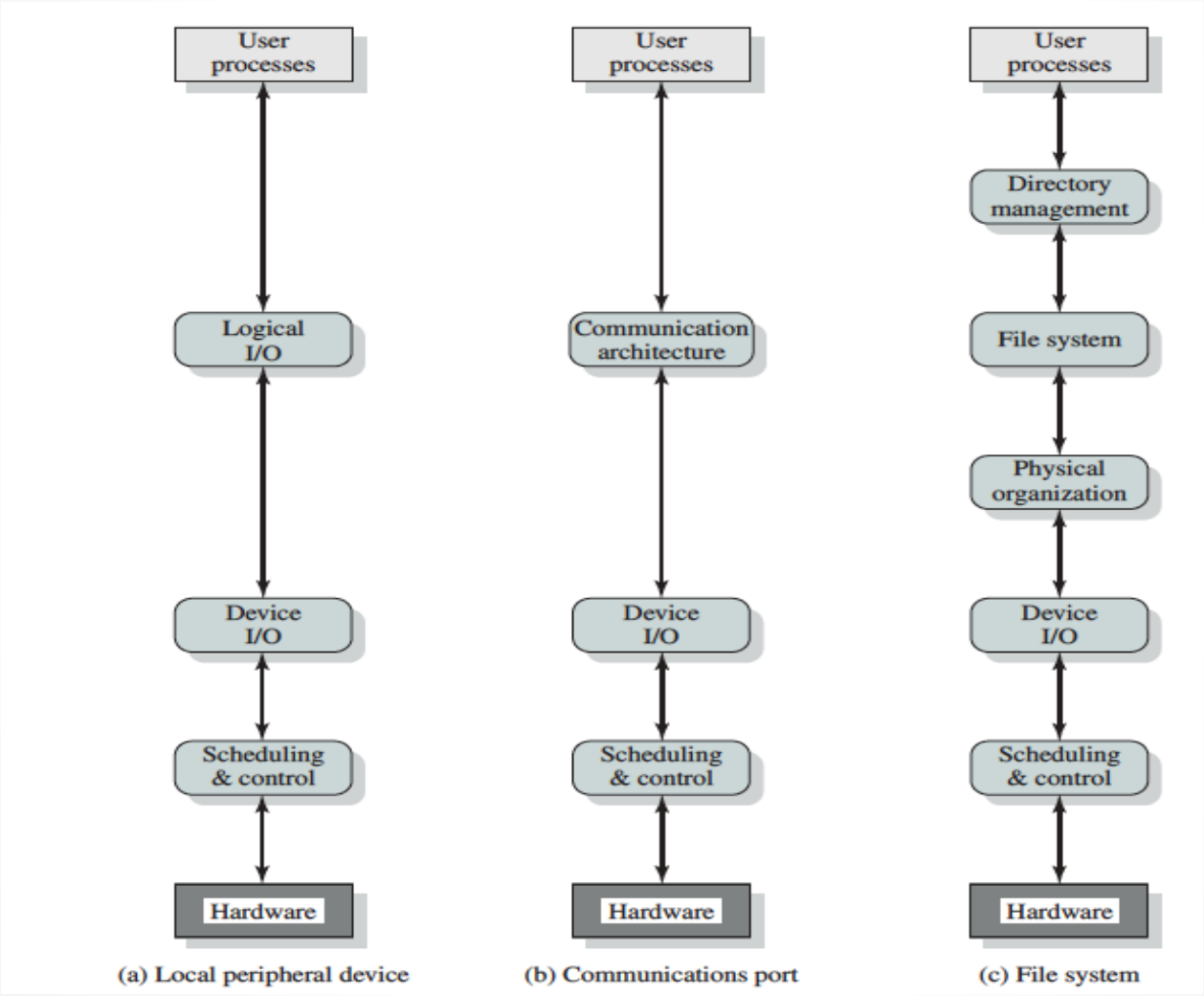
Objectives

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

Contents

- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Productivity and performance

A model of I/O organization

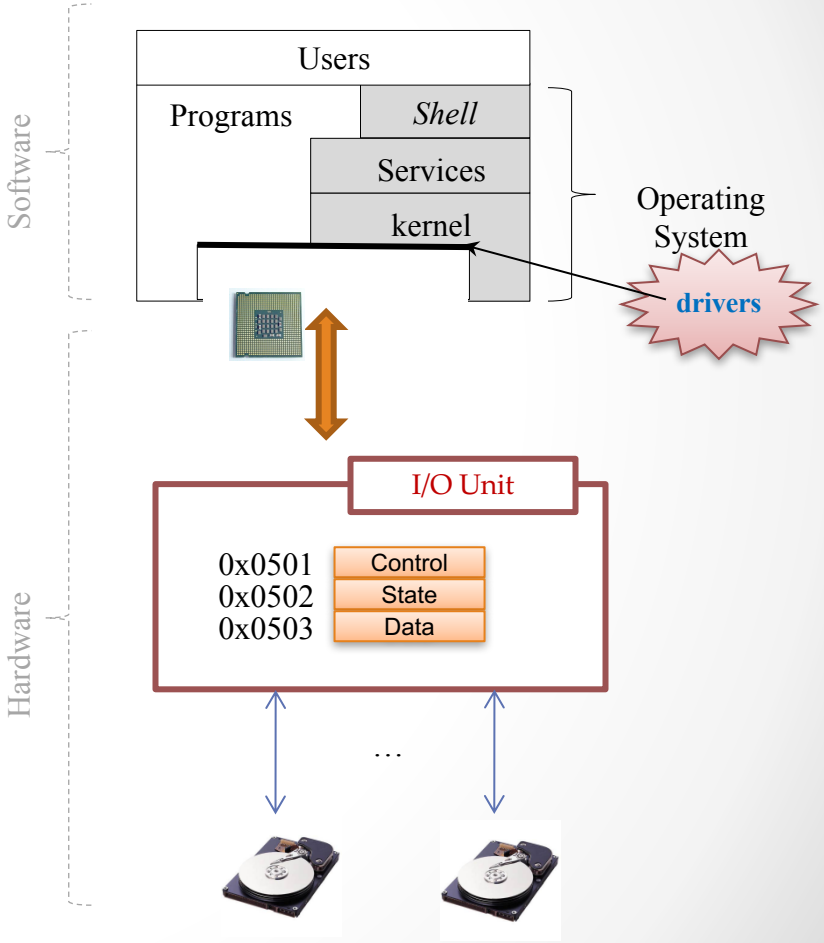


Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks

Drivers

- Part of the OS devoted to interact with the all controllers of the I/O devices (hardware)
- Strong OS dependency:
  - Drivers of an OS are difficult to be reused
- Dynamic components:
  - Many new drivers



Drivers importance

- Linux kernel statistics (2008-2009):
  - 9,2 millions lines of code.
  - 10% increment each year:
  - 55% of code are drivers:
    - Software part of the OS that the CPU executes to cooperate with the associated device
    - Kernel level code
  - Kernel source is only 5%. Another 40% is for adaptation to architectures, network code, services, etc.

http://cityblogger.com/archives/2008/06/16/linux-kernel-stats

Quiz

- What is **not** true about a device driver:

a) It typically runs in privileged mode

b) You have them for character or block devices.

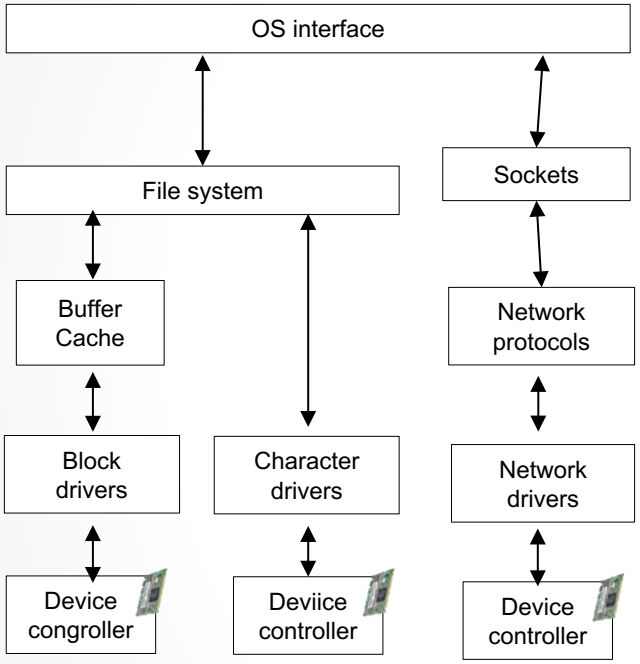
c) They run only when the system is booted.

d) In UNIX there is a uniform access interface for device drivers.

Contents

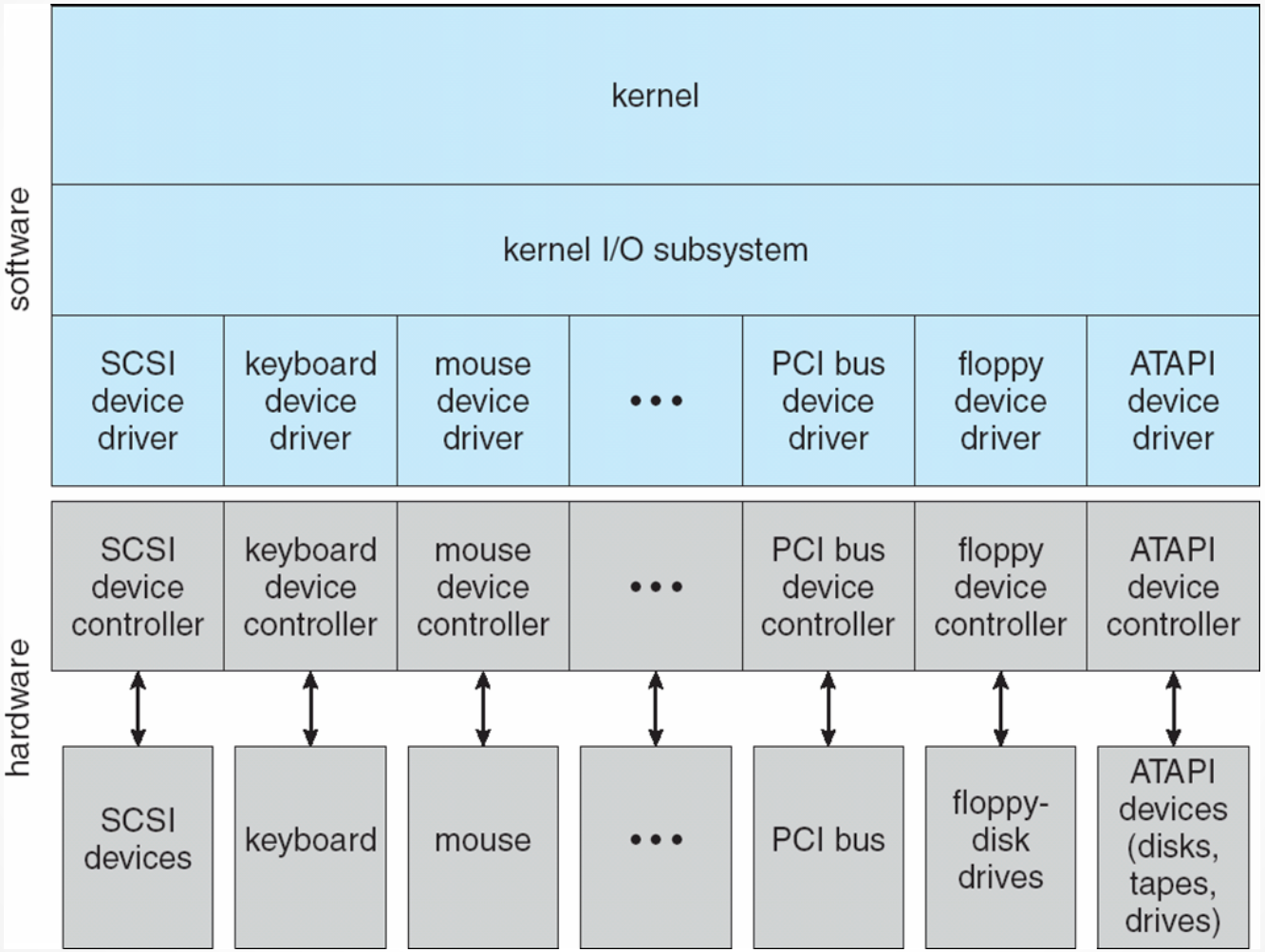
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Productivity and performance

I/O goals



- To offer a simplified logical vision for:
  - Remainder of the OS
  - Users
- Optimizing I/O
- Facilitating peripheral management
- Facilitating the inclusion of new devices

A Kernel I/O Structure



Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Characteristics of I/O Devices

- Subtleties of devices handled by device drivers
- Broadly I/O devices can be grouped by the OS into
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets
- For direct manipulation of I/O device specific characteristics, usually an escape / back door
  - Unix `ioctl()` call to send arbitrary bits to a device control register and data to device data register

Quiz

- ioctl in the Linux system is :

a) A general purpose function used only for sending data to the device.

b) A program of the operating system to mount/unmount I/O device on the system.

c) A system call for issuing operations to any device. d) A I/O control process of the kernel.

Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - **Raw I/O**, **direct I/O**, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA
- Character devices include keyboards, mice, serial ports
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing

Network Devices

- Varying enough from block and character to have own interface
- Unix and Windows NT/9*x*/2000 include **socket** interface
  - Separates network protocol from network operation
  - Includes `select()` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

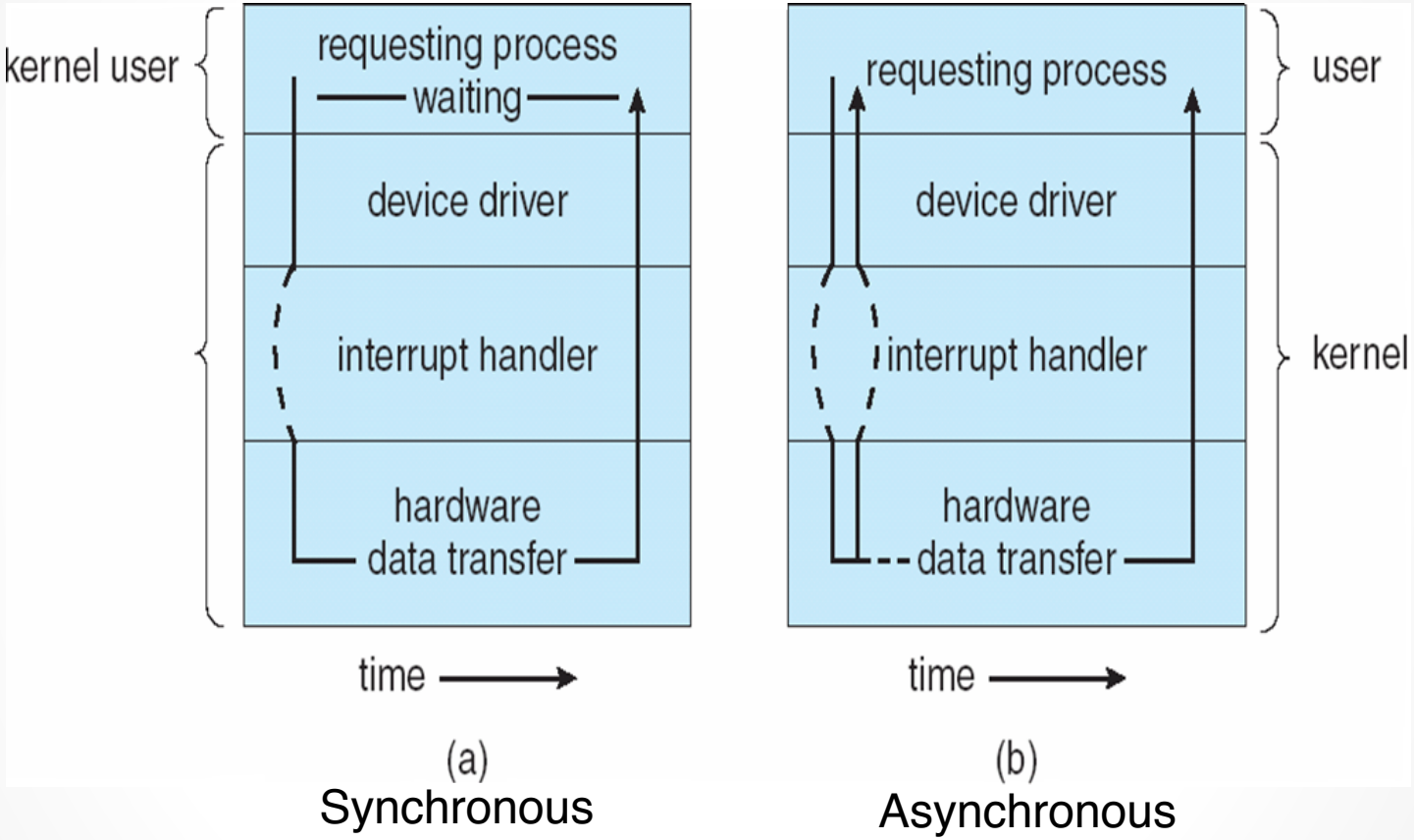
Clocks and Timers

- Provide current time, elapsed time, timer
- **Programmable interval timer** used for timings, periodic interrupts
  - Scheduler
  - Periodic flushing of caches
  - Cancel slow network operations
  - Generally: time-scheduled events
- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers

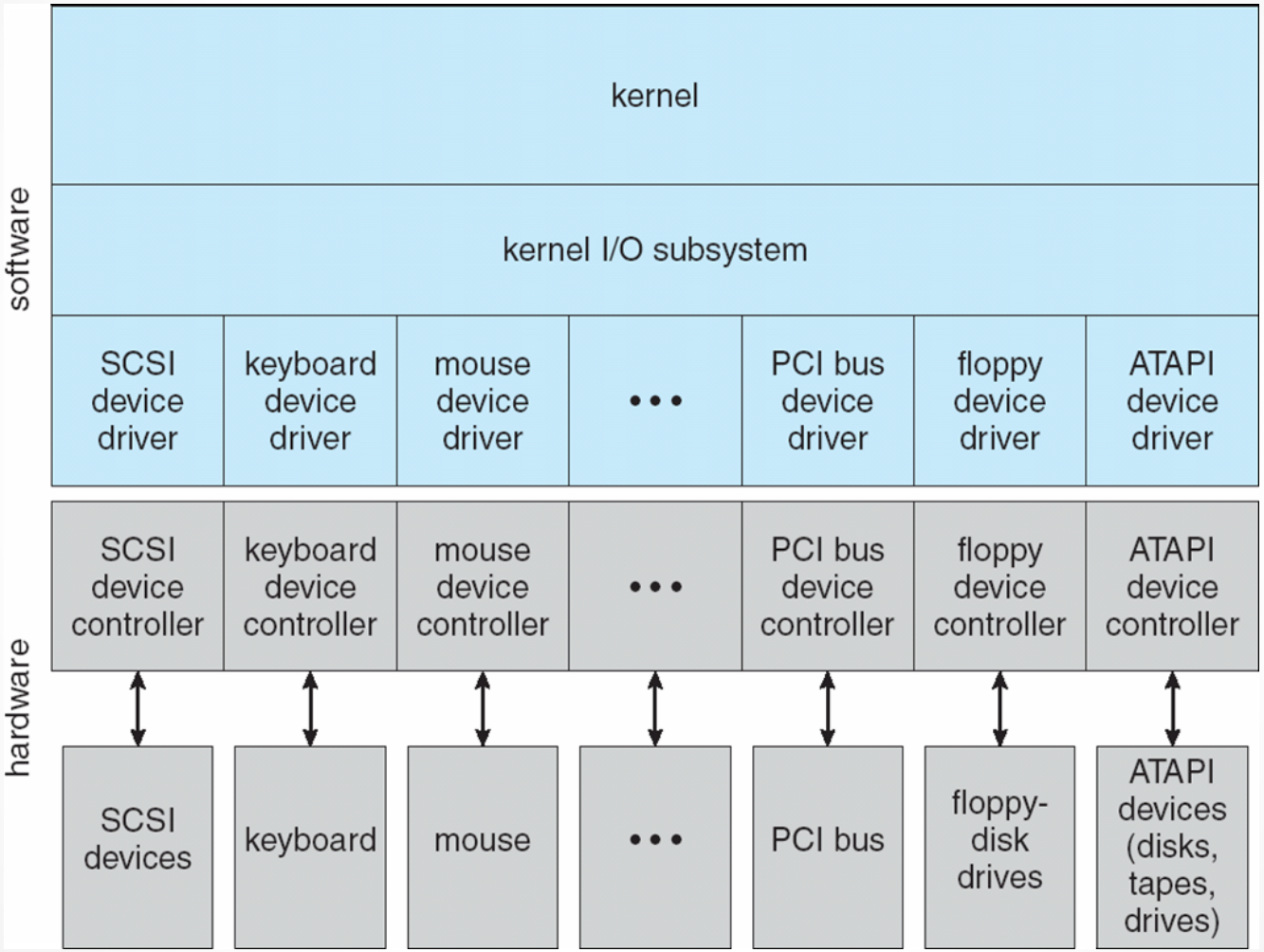
Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
  - Select
- **Asynchronous** - process runs while I/O executes
  - Some consider the same as nonblocking
  - Some consider that the main difference is:
    - Nonblocking just returns the information that is available and an error if it the full operation can not be finished
    - Asynchronous starts the operation and provides a mechanism for completion detection (interrupt, check)

Two I/O Methods



Kernel I/O Subsystem

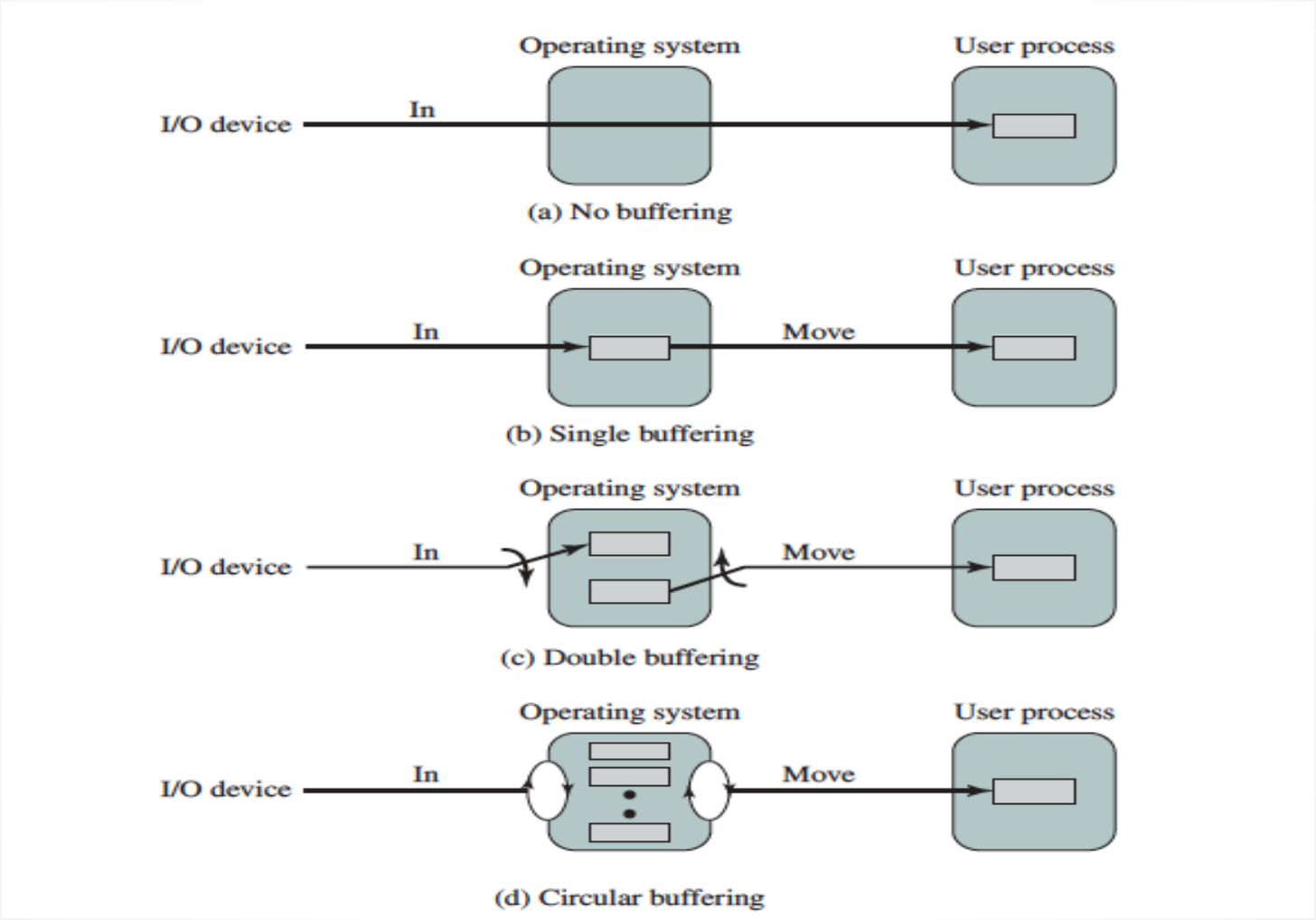


Kernel I/O Subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness

- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain "copy semantics"
    - E.g. Copy data from application to kernel buffers before transfer to disk or network

I/O buffering schemes (input)



Quiz

- What is not a use of a buffer?

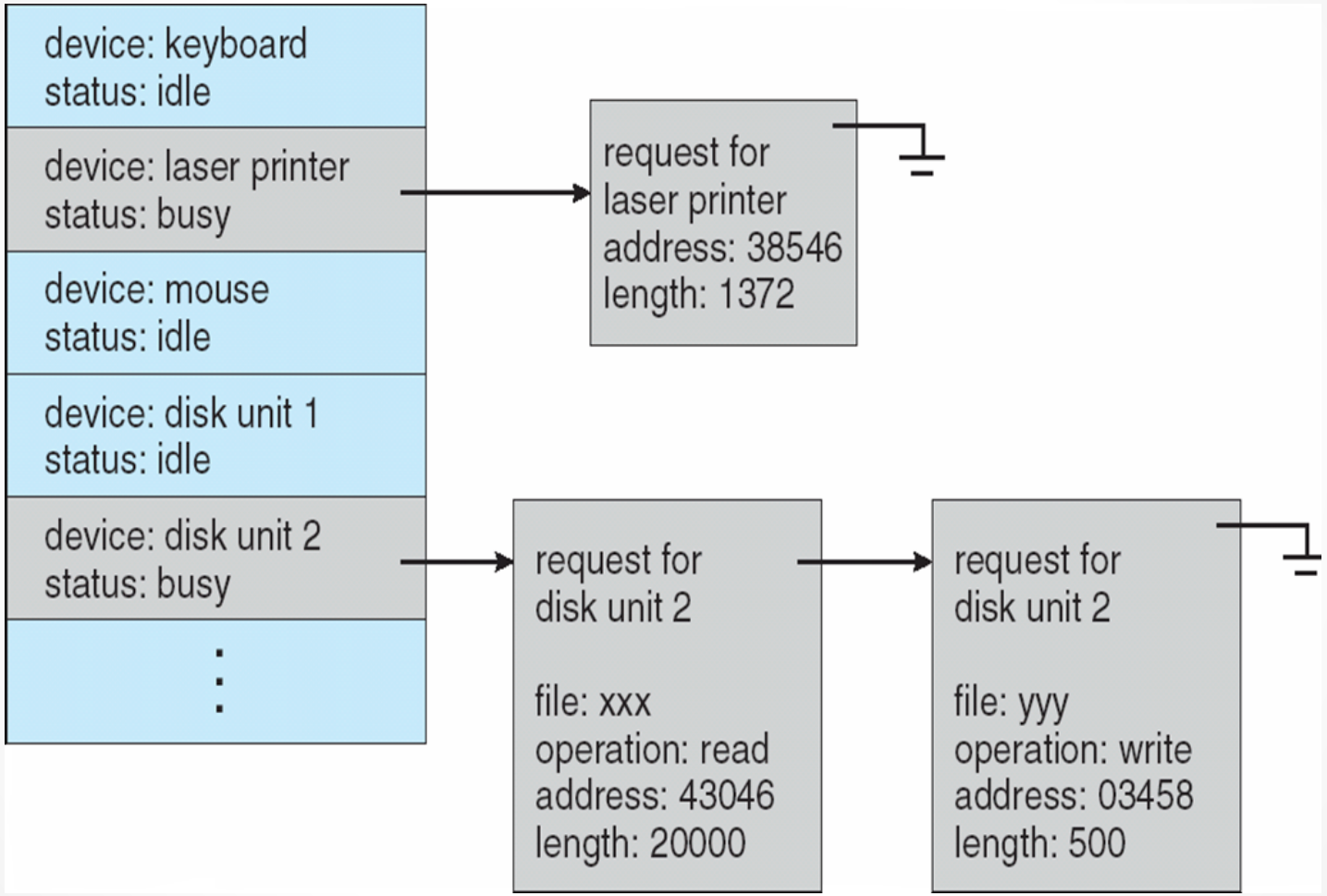
a) For adapting the speed mismatch between a producer and a consumer

b) For merging data targeting the same destination

c) For transferring data between devices with different transfer unit sizes

d) For data consistency

Device-status Table



Kernel I/O Subsystem

- **Caching** - fast memory holding copy of data
  - Always just a copy on faster storage
  - Key to performance
  - Difference: Buffers may contain the only copy

- **Spooling** - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing

- **Device reservation** - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock

Quiz

- Which of the following may hold the only existing copy of a data item.

a) only a cache

b) only a buffer

c) either a cache or a buffer

d) neither a cache or a buffer

Quiz

- A spool is a buffer that can accept interleaved data streams from various devices

a) True

b) False

Quiz

- Choose the affirmation which is incorrect.

a) All I/O instructions defined to be privileged

b) I/O must be performed via system calls

c) Memory-mapped and I/O port memory locations must be protected

d) I/O can be performed only at user level

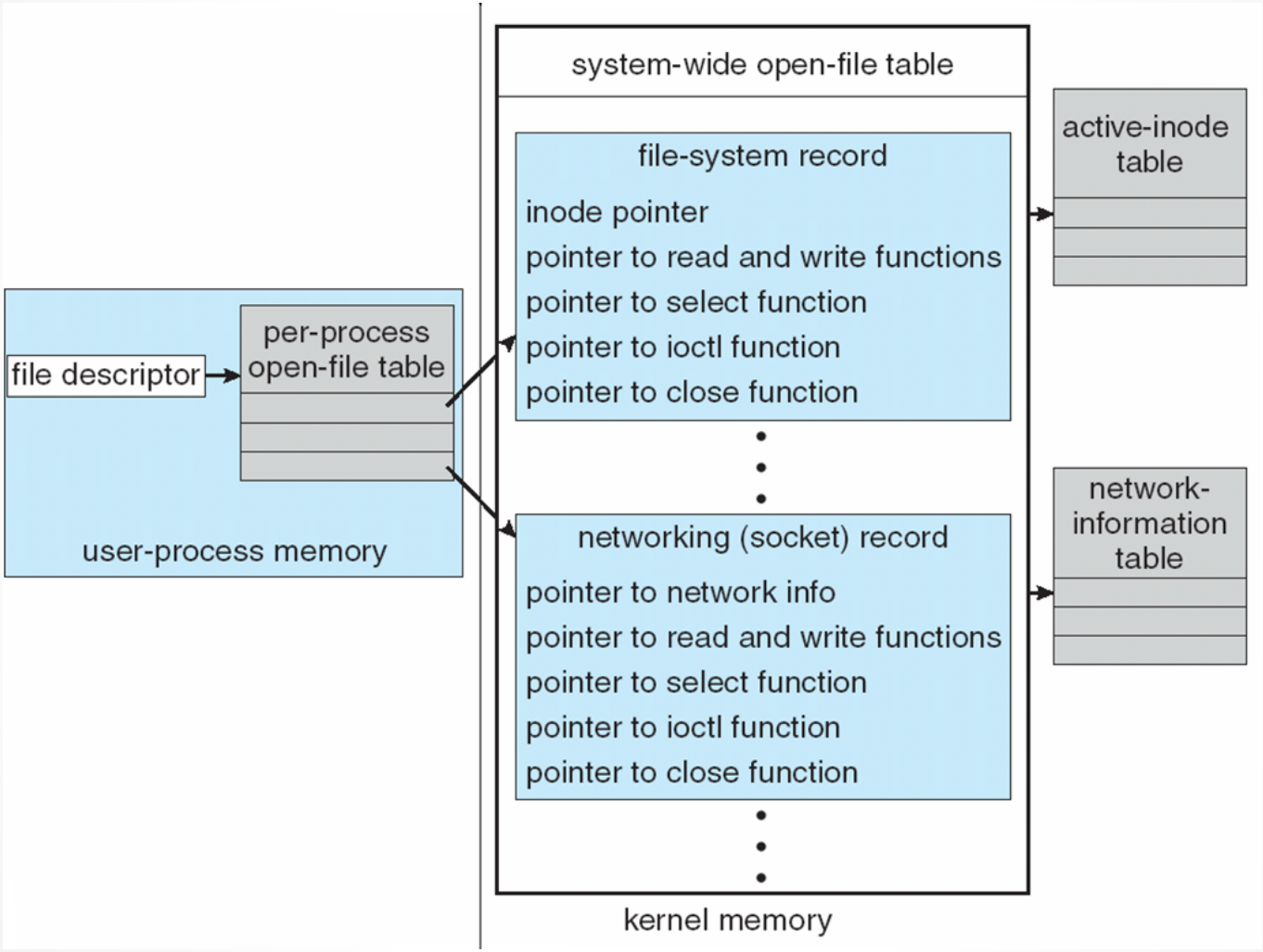
I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, "dirty" blocks
- Some use object-oriented methods and message passing to implement I/O

UNIX I/O Kernel Structure

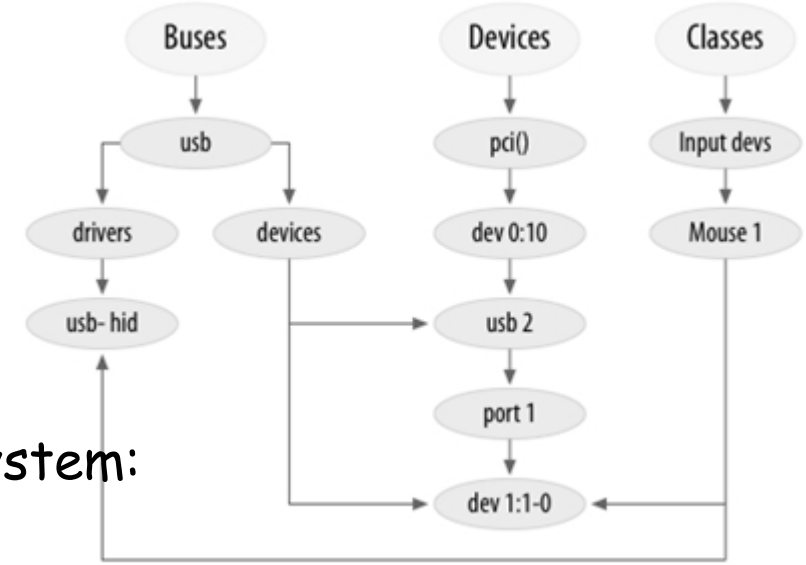


Linux drivers identification

- Device identification:
  - As files : `/dev/xxxx`
  - Dev ID: *Major number* (driver)+ *minor number* (device)
    - brw-r----- 1 root operator 1, 1 2 mar 18:52 disk0s1
    - crw-rw-rw- 1 root wheel 4, 0 2 mar 18:52 ttyp0
  - Driver management:
    - mkdev (obsolete): script to create possible files
    - devfs (obsolete): File system with all possible drivers
    - udev: dynamic driver management (*hot-plug/unplug*, *triggers*, etc.)

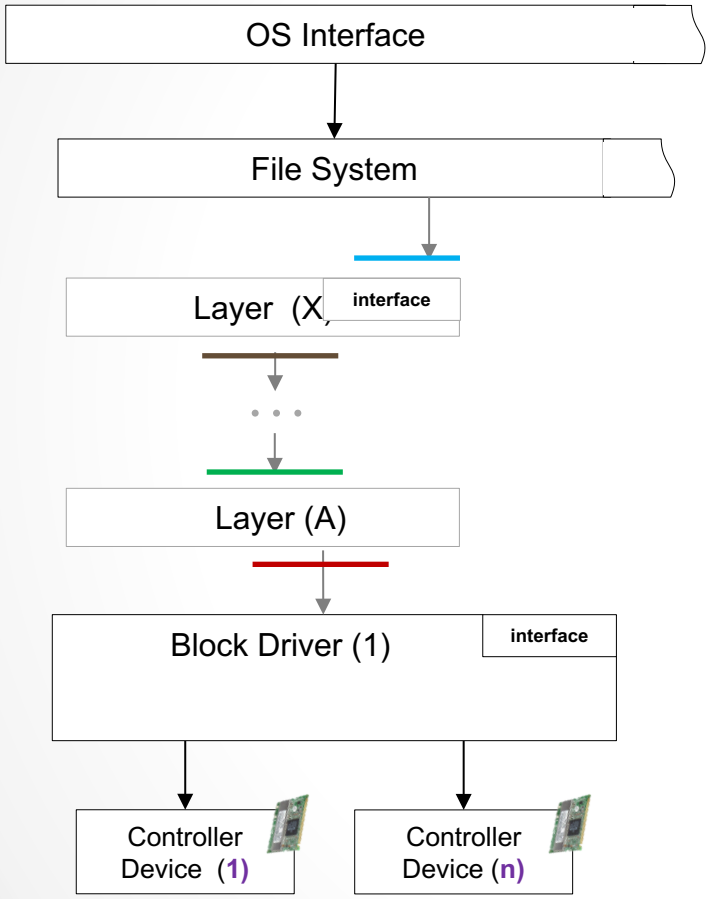
Linux drivers hierarchy

- Shown in the figure:
  - Buses in lower level
  - Devices in middle level
  - Classes in higher level



- Access through sysfs pseudo file system:
  - /sys/block: block devices (any bus)
  - /sys/bus: system buses
  - /sys/devices: devices organized by buses
  - /sys/class: classes of devices (audio, network, ...)
  - /sys/module: drivers registered in the kernel
  - /sys/power: energy state management
  - /sys/firmware: firmware management (in some devices)

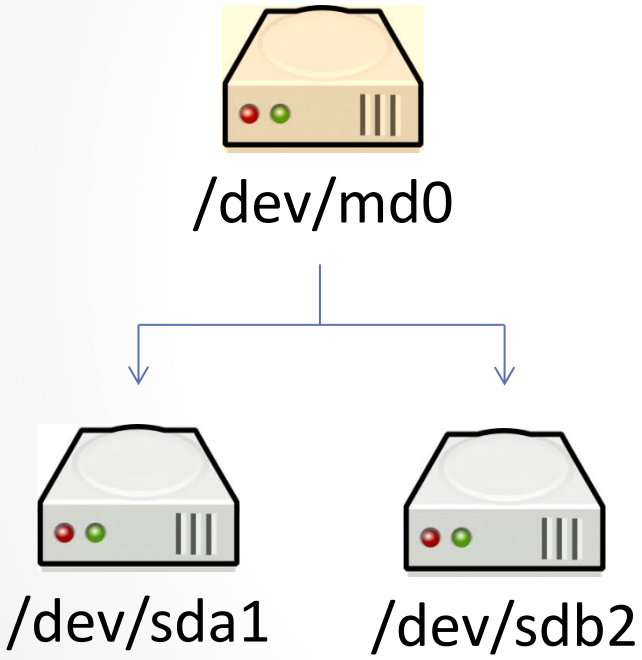
Drivers hierarchy: layered stacking



- Extended services:
  - Module extending a driver with new functionality.
  - May be stacked.

- At least two interfaces:
  - Service interface
    - System call interface
    - Interface of an upper extended service
  - Interface of the resource used:
    - Interface of a driver
    - Interface of an Extended service

Extended services Linux



- Example of extended service:
  - md (*multiple disks*)
- Joins several HD, or partitions (or volumes) in a single virtual disk.
  - `mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sda1 /dev/sdb1`

Contents

- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Productivity and performance

I/O Requests to Hardware Operations

- Consider reading a file from disk for a process:
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process

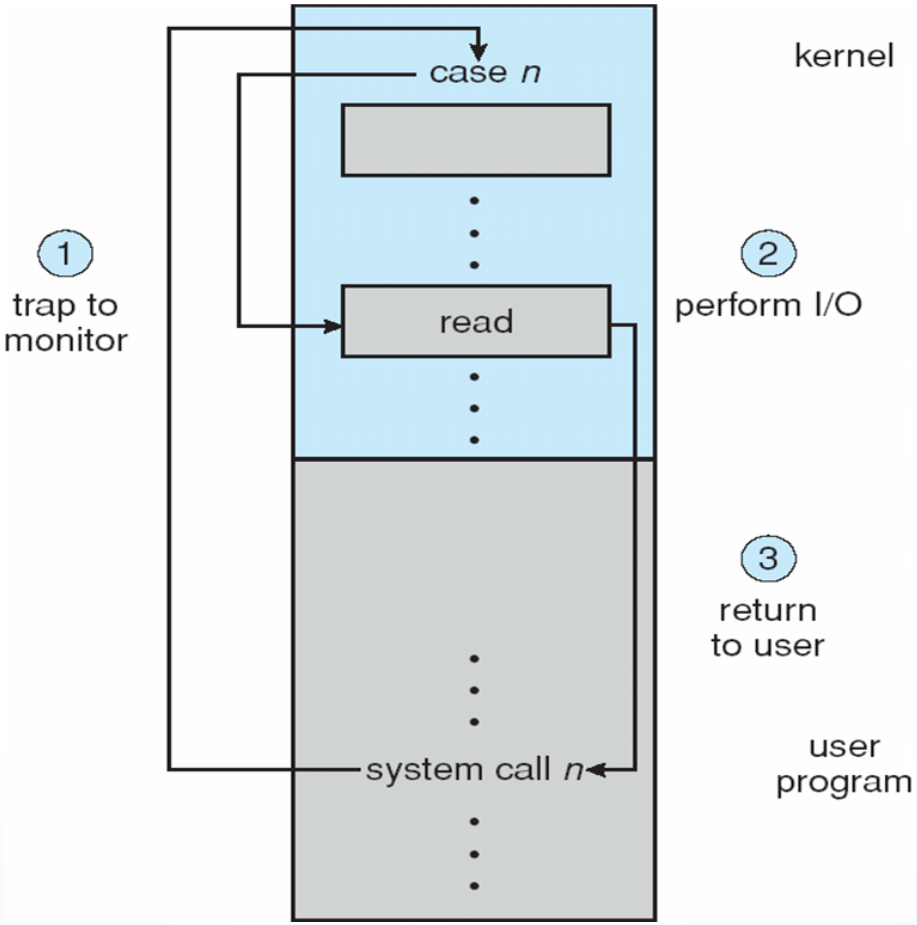
System Call Interface

Basic I/O modes in Linux

	Blocking	Non-blocking
Synchronous	Read/write	Read/write (O_NONBLOCK)
Asynchronous	I/O multiplexing (select/poll)	AIO

http://www.ibm.com/developerworks/linux/library/l-async/

Use of a System Call to Perform I/O



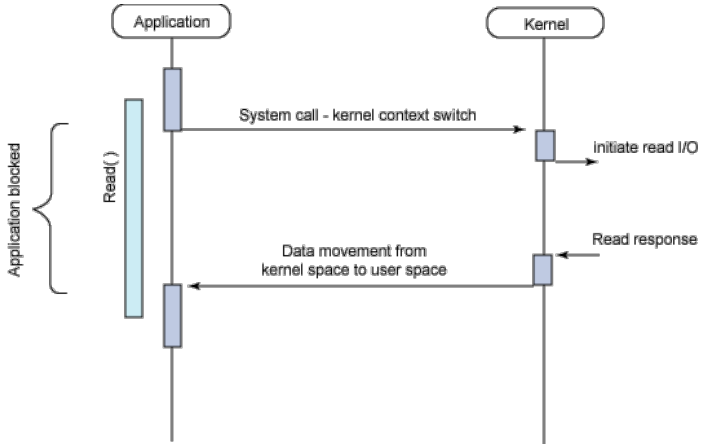
System Call Interface

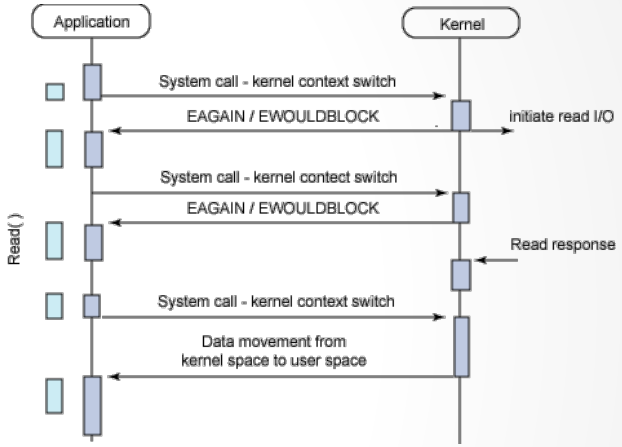
Basic I/O modes in Linux

Blocking
(notification)

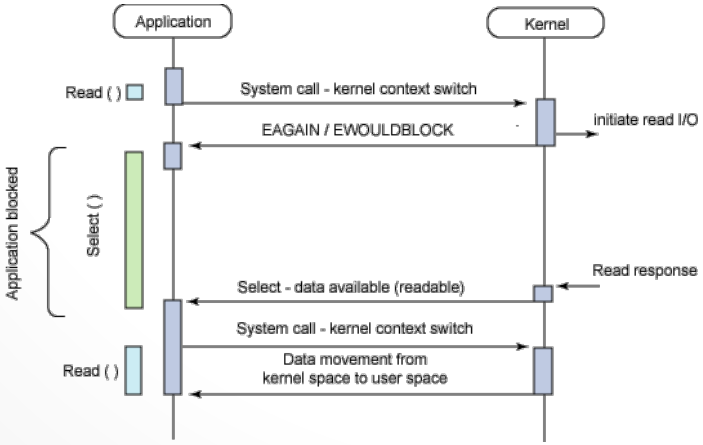
NON-blocking
(notification)

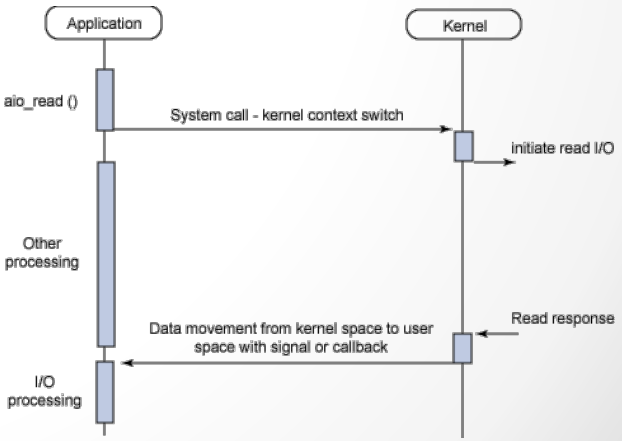
Sync
(request)





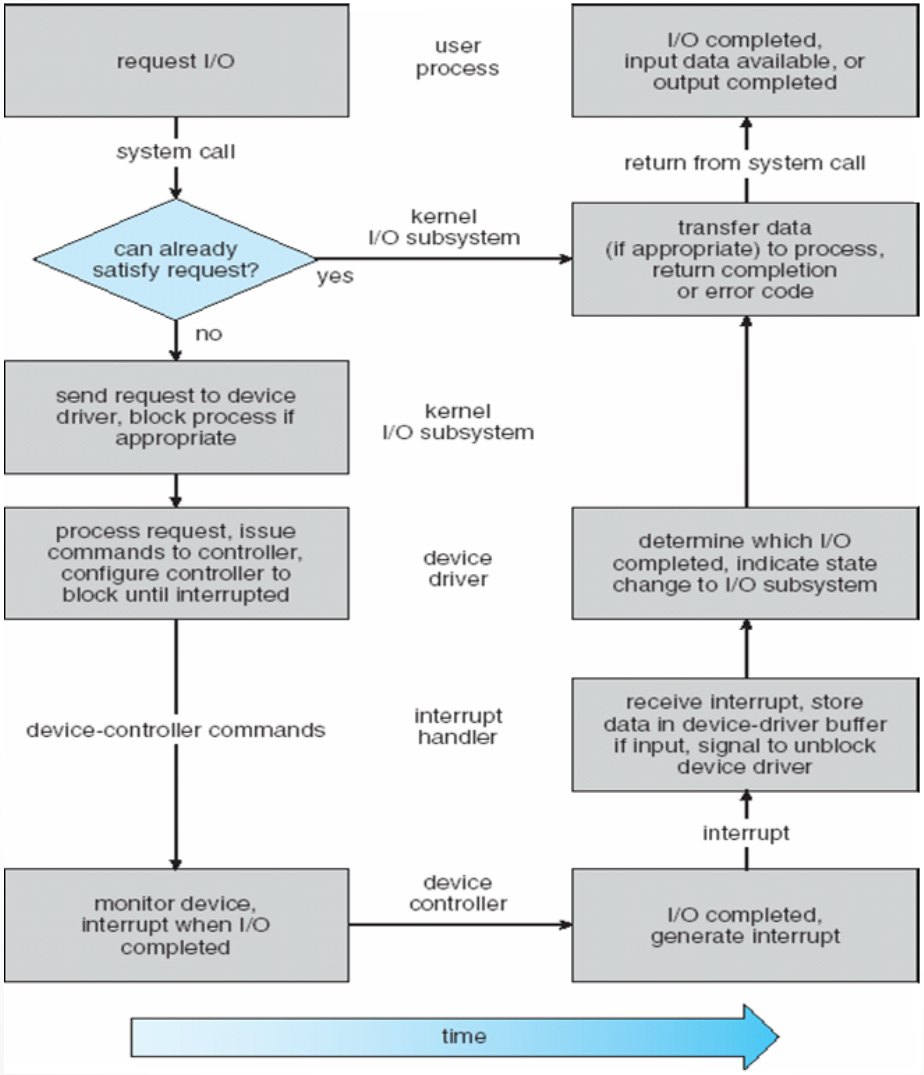
Async
(request)





http://www.ibm.com/developerworks/linux/library/l-async/

Life Cycle of An I/O Request



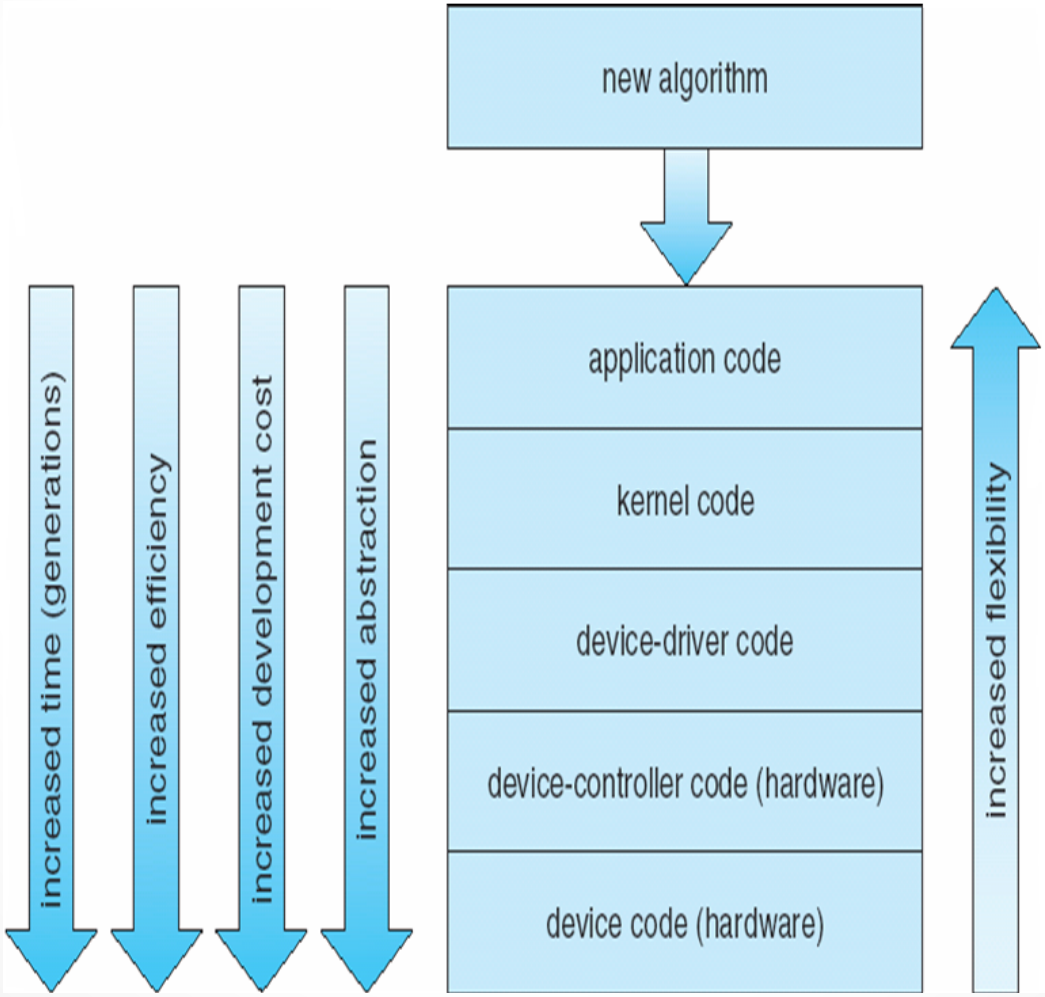
Error Handling

- OS can recover from disk read, device unavailable, transient write failures
  - Retry a read or write, for example
  - Some systems more advanced – Solaris FMA, AIX
    - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

Contents

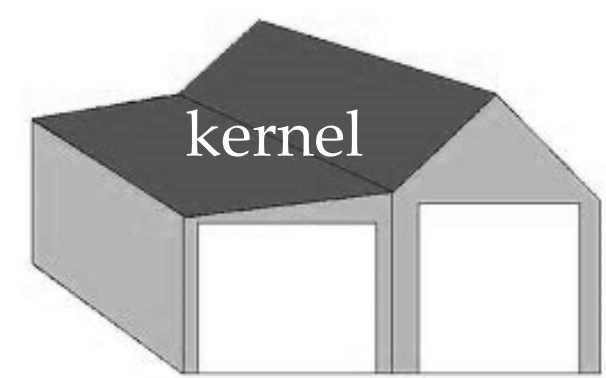
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Productivity and performance

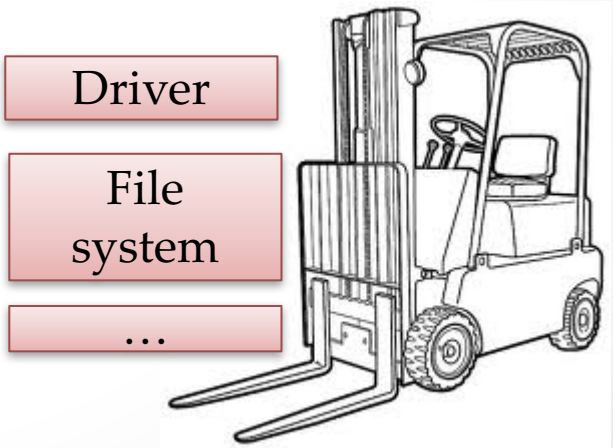
Device-Functionality Progression



Modules for including drivers into the kernel

- Modules are a mechanism used to include new device drivers into the Linux kernel
  - Dynamically linked to the kernel without recompiling the kernel
- Also used to include more functionality:
  - File systems, network protocols, new system calls, etc.





Types of modules

- Device drivers
- Filesystem driver (one for ext2, MSDOS FAT16, 32, NFS)
- System calls
- Network Drivers
- TTY line disciplines. special terminal devices.
- Executable interpreters.

General implementation steps

1. Understand the device characteristic and supported commands.
2. Map device specific operations to unix file operation
3. Select the device name (user interface)
   - Namespace (2-3 characters, /dev/lp0)
4. Select a major number and minor (a device special file creation) for VFS interface
   - Mapping the number to right device sub-routines
5. Implement file interface subroutines
6. Compile the device driver
7. Install the device driver module with loadable kernel module (LKM)
8. or Rebuild (compile) the kernel

General file operations

```
struct file_operations {
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
                int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *);
        int (*fasync) (int, struct file *, int);
        int (*check_media_change) (kdev_t dev);
        int (*revalidate) (kdev_t dev);
        int (*lock) (struct file *, int, struct file_lock *);
};
```

Implementation

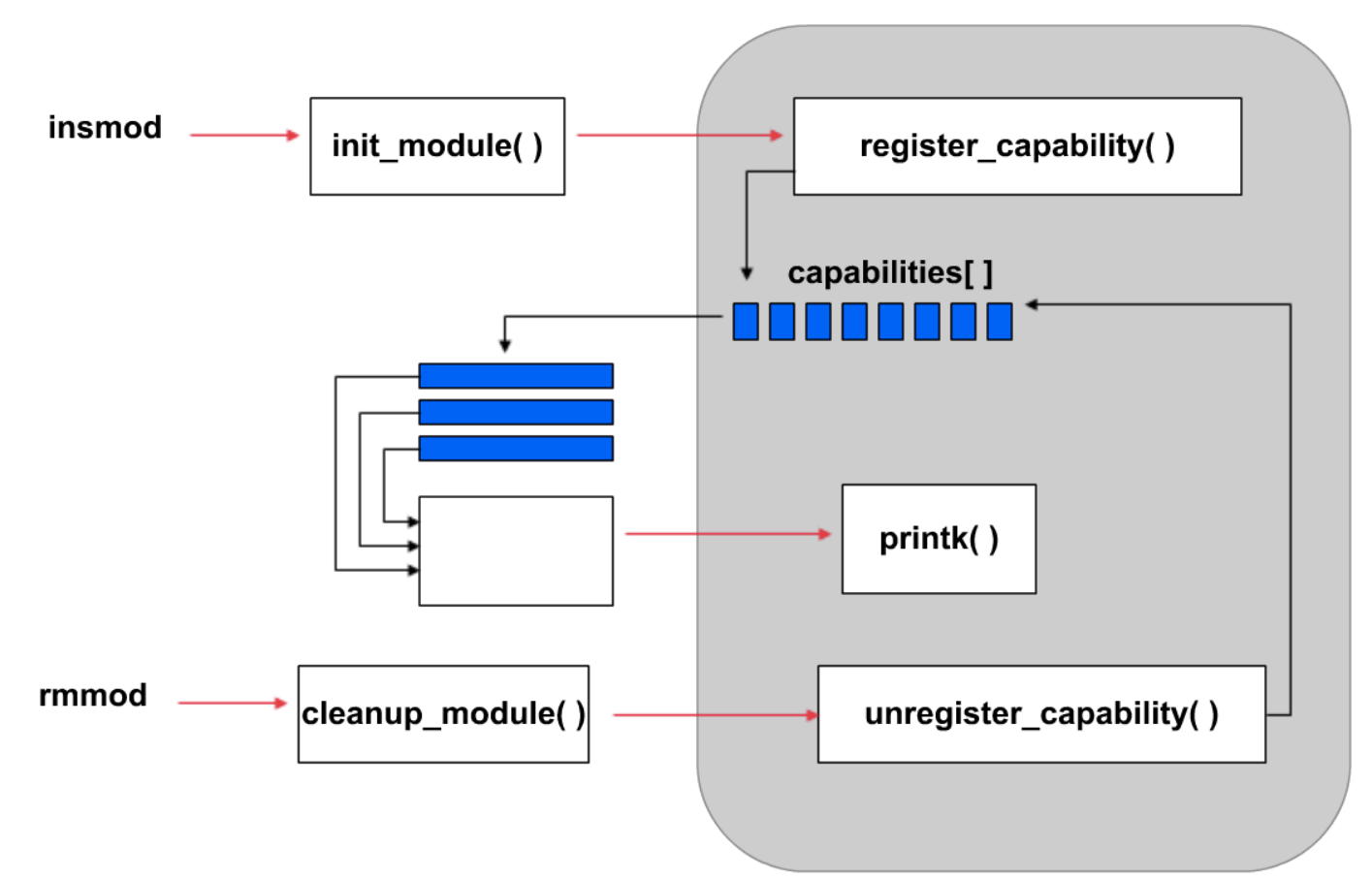
- Create a device
  - Make a special file
    % mknod /dev/device_name c major minor
  - "c" for char devices, "b" for block devices.
  - register_chrdev (major_num, "name", &file_operation_struct)
  - register_blkdev (major_num, "name", *file_operation_struct)

- Assuming that the device name is Xxx
  - Xxx_init() initialize the device when OS is booted
  - Xxx_open() open a device
  - Xxx_read() read from kernel memory
  - Xxx_write() write
  - Xxx_release() clean-up (close)
  - init_module()
  - cleanup_module()

Example: Hello World

```
#define MODULE
#include <linux/module.h>
int init_module(void) {
  printk( “<1>Hello, world!\n” );
  return 0;
}
void cleanup_module(void) {
  printk( “<1>Goodbye cruel world ☹\n” );
}
```

A module in the kernel



Managing the modules

- compile

  -Wall -DMODULE -D__KERNEL__ -DLINUX -DDEBUG -I /usr/include/linux/version.h -I/lib/modules/`uname -r`/build/include

- Install the module

  %insmod module.o

- List the module

  %lsmod

- Remove an LKM from the kernel
  - %rmmod module.o
- Display contents of .modinfo section in a module.
  - %modinfo module.o

- Insert or remove an LKM or set of LKMs intelligently. For example, if you must load A before loading B, Modprobe will automatically load A when you tell it to load B.
  - modprobe module.o

Performance

- I/O a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Quiz

- The I/O performance can be improved by:

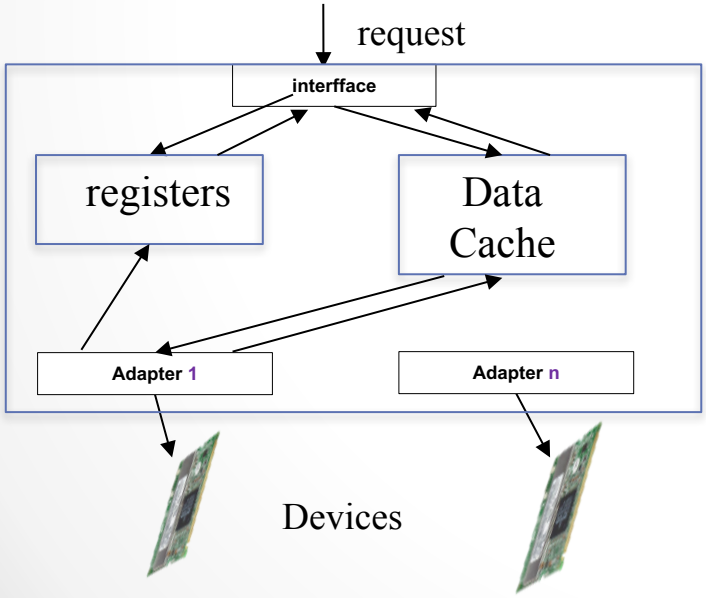
a) Reducing the number of context switches
b) Reducing the data copying
c) Using large transfers
d) Increasing the number of interrupts
e) Using DMA

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O performance for highest throughput
- Move user-mode processes / daemons to kernel threads

Devices cache. E.g. disk read

- Copy of data recently requested.
- The largest the better, but it consumes memory …
- Cache policy critical to have good hit ratio (LRU, LFU, …)
- Specially well suited for block devices. Eg. Disks
- Goal: high hit ratio



Device controler manager:

Store control data in registers
Look for data in cache
if (Data found)
    Insert data in memory buffer
Else
    Find device to sent request to
    Read data from device to cache
    Insert data in memory buffer
Record I/O status in registers
Return

Grupo ARCOS

Operating Systems Design
Grado en Ingeniería Informática

Universidad Carlos III de Madrid