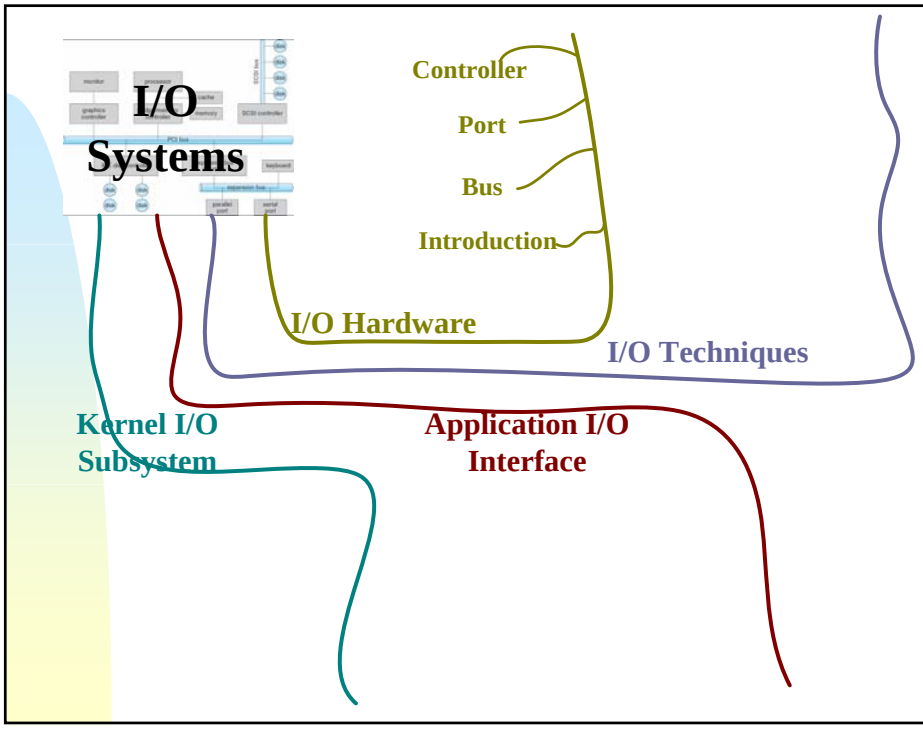
Module 11: I/O Systems

Reading: Chapter 13

Objectives

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



Types of I/O Devices

- For human usage
  - printers
  - monitors
  - keyboard…
- For the machine
  - disks, tapes…
- For communication
  - modems, network cards...

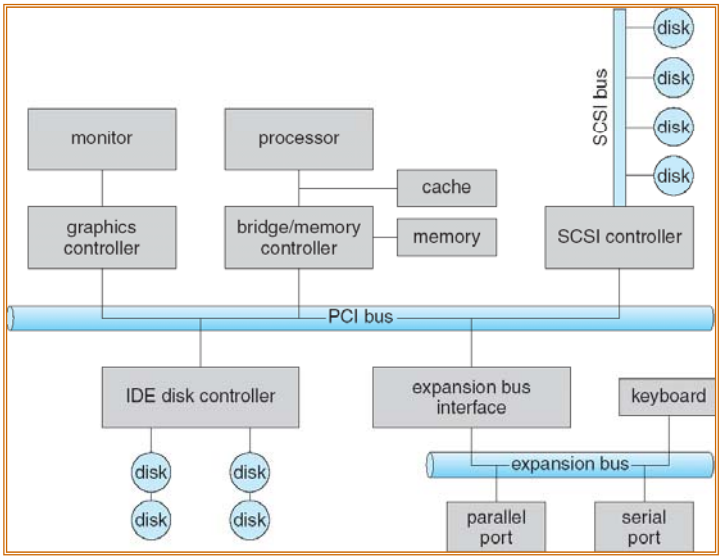
Main Characteristics

- Speed (data rate)
  - enormous differences
- Unit of data transfer
  - Byte stream (ex: keyboard)
  - Blocks (ex: disk)
- Data representation
  - Different data encoding schemes adopted (character codes, parity conventions...)
- Error conditions
  - Nature of errors and responses vary a lot
- Still, we want to have as simple and unified view of them as possible

I/O Hardware

- I/O Port
  - **Connection point for communicating with device**
- Bus
  - **wires + protocol**
  - **Shared connection point**
- Controller
  - **Electronics controlling port/bus/device**
  - **Has control registers**
    - **i.e. data-in, data-out, status, control**
  - **+ other stuff (buffers, …)**
- **Devices have addresses**
  - **Direct I/O instructions**
  - Memory-mapped **I/O**

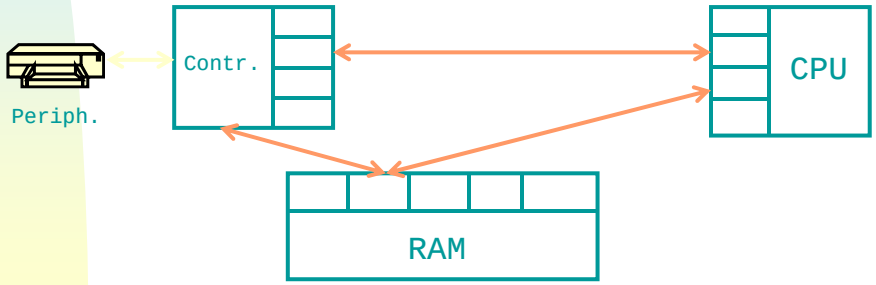
A Typical PC Bus Structure



PCI: Peripheral Component Interconnect

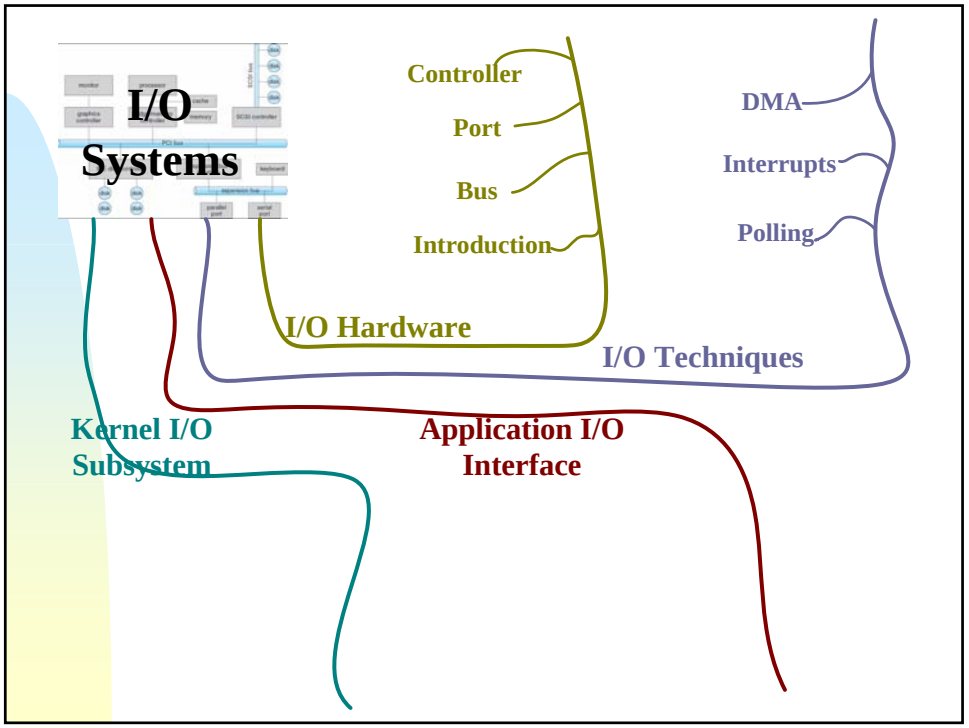
Communication between CPU and I/O Controllers

- **Two basic techniques:**
  - **CPU accesses controller registers.**
  - **CPU and controllers can exchange data via zones in main memory**
  - **Combination of both techniques**



Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)



I/O using Polling

- CPU reads/writes the I/O Port/Controller registers
- Handshaking between the CPU and the controller
  - Ready bit in a register indicates if the controller is busy (=1) ore can accept a new command (=0).
- Busy-waiting/polling necessary when ready bit indicates device is busy
  - Low overhead if ready bit becomes 0 soon
  - But lots of waste (busy waiting) if the ready bit remains at 1 for a long time
- Solution?
  - interrupts

Interrupts

Idea: **don't poll the device, but let the device tell the CPU when it is ready**

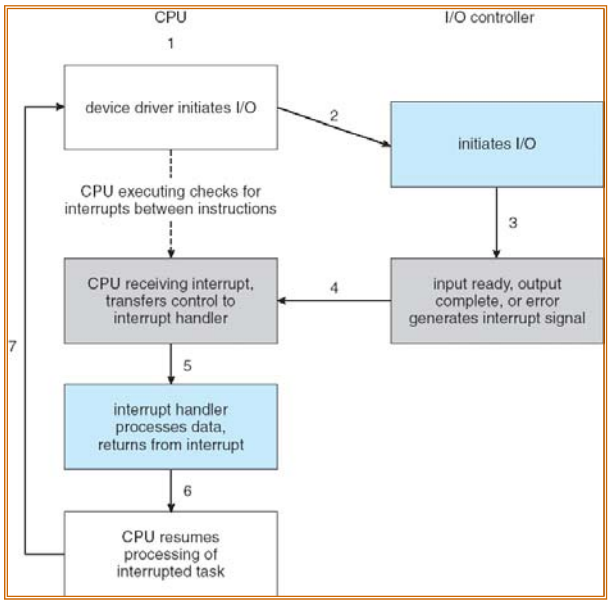
How the device signals to the CPU?

- Interrupt-request line **triggered by I/O device**

How does the CPU know which device was involved?

- **Can poll all devices**
- **From the interrupt #**
  - **Index to the** Interrupt Vector **table, pointing to** Interrupt Handlers

Interrupt-Driven I/O Cycle



Intel Pentium Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Interrupts

Interrupt attributes

- Maskable/non-maskable
- **Interrupt priorities**

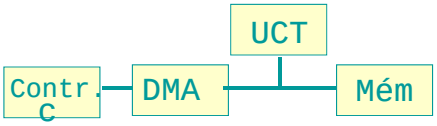
Interrupt uses

- **Interrupt-based I/O**
- **Handling exceptions**
- **Invoking OS kernel (system calls)**
- **Managing flow of control**

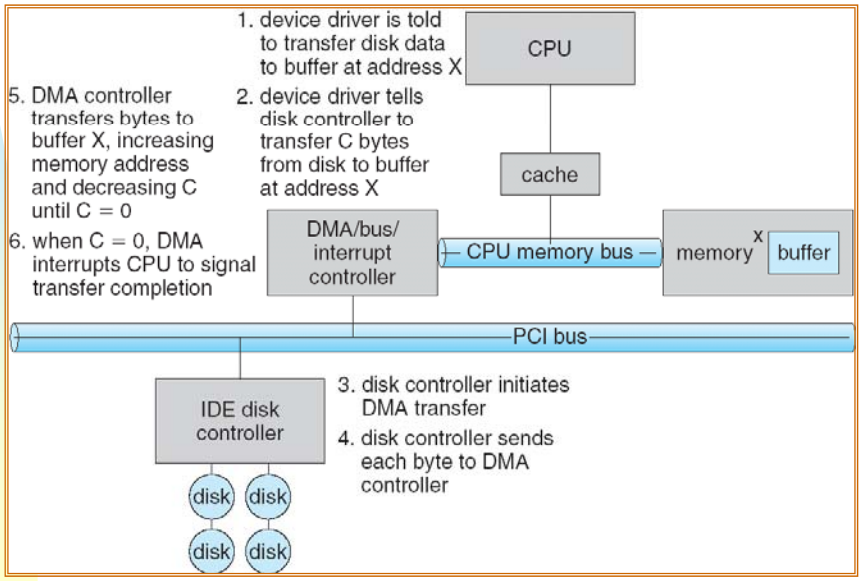
Direct Memory Access (DMA)

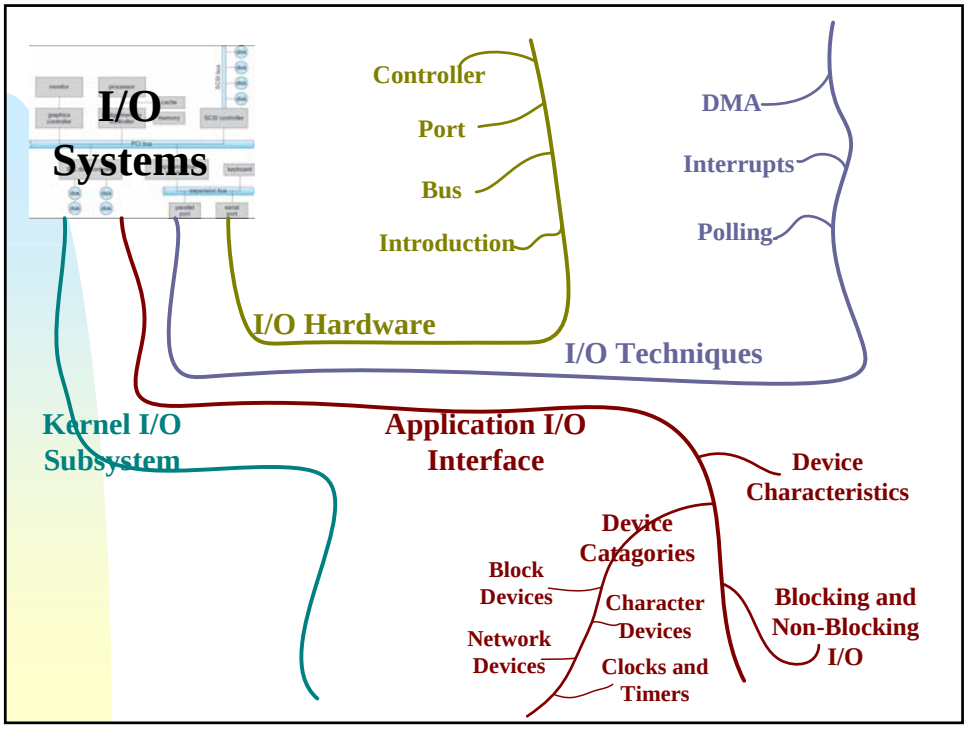
- **In systems with no DMA, the CPU is involved in the transfer of each byte.**

- **DMA is used to exclude the involvement of the CPU, particularly for large I/O.**
  - **Requires a special controller**
  - **The controller accesses memory directly**
  - **DMA still slows the CPU, since it uses the bus to the memory which prevents the CPU from using it**
  - **But much less than having the CPU more involved in the I/O.**



Direct Memory Access – 6 Steps





Application I/O Interface

- the OS must perform the I/O on behalf of the user processes
- ideally, all I/O devices would be accessed in a uniform way
  - OS hides hardware differences behind a uniform API
- in reality, the I/O devices are way too different
  - block or stream oriented
  - only read, only write, read/write
  - sequential/random access
  - synchronous/asynchronous
  - vastly different transfer rates
  - sharable/dedicated

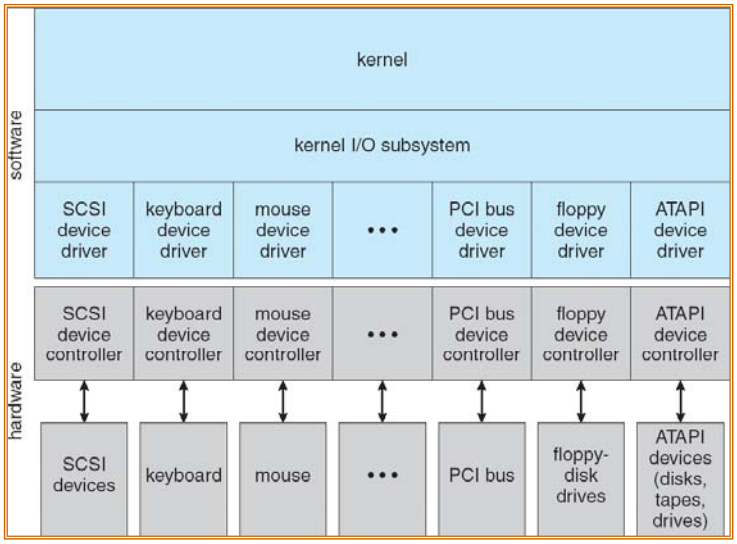
Characteristics of I/O Devices

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

Application I/O Interface

- Modular, hierarchical, multi-layered model is generally used
- The I/O system encapsulates the behaviour of peripherals into major device categories:
  - Block I/O or Character-stream I/O
  - Memory-mapped file
  - Network sockets
- Additional standard devices
  - Clock, timer
- The driver layer masks the differences between the various peripherals
- Backdoor function for sending arbitrary commands to controllers
  - If the device is very specialized and none of the methods from the interface fit well
  - To fully exploit the device's additional functionality
  - Unix - ioctl(fd, command, args)
- The developer of drivers must create a driver for each type of operating system

A Kernel I/O Structure



Block and Character Devices

- **Block devices include disk drives**
  - **Commands include `read`, `write`, `seek`**
  - **Managed by file-system**
  - **Raw I/O can be used (i.e. databases)**
  - **Compromise: direct I/O**
    - **FS, but not buffering/locking**
  - **Memory-mapped file access possible**
- **Character devices include keyboards, mice, serial ports**
  - **Commands include `get, put`**
  - **Libraries layered on top to 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)**

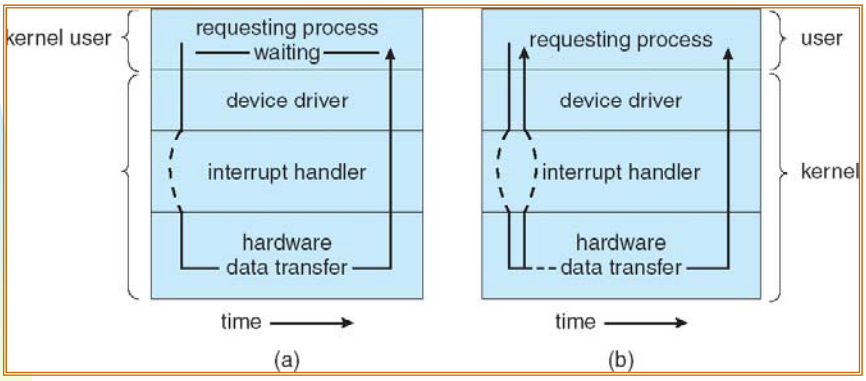
Clocks and Timers

- **Provide current time, elapsed time, timer**
- Programmable interval timer **used for timings, periodic interrupts**
- **`ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers**
- **Other calls such as `sleep()`, `alarm()`, `getitimer()`, and `setitimer()` provide timing services**

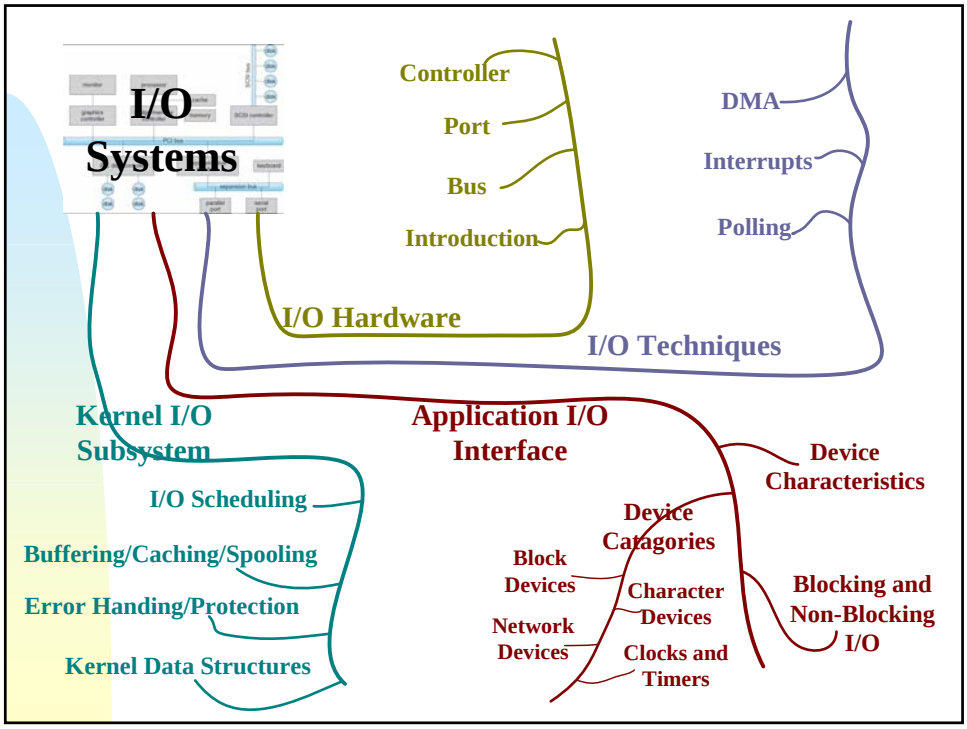
Blocking and Nonblocking I/O

- 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)**
  - **Returns quickly with count of bytes read or written**
  - **Can be implemented using multithreading and blocking I/O**
- Asynchronous - **process runs while I/O executes**
  - **Difficult to use**
  - **I/O subsystem signals process when I/O completed**

Two I/O Methods



Synchronous Asynchronous



I/O Management Objectives

Efficiency

- I/O operations are much slower than processor operations
  - quite often I/O is the bottleneck
  - any inefficiency in I/O directly impacts the whole system performance
- what can be done?
  - use DMA or interrupt driven I/O
  - schedule the processes so that the I/O devices are fully utilized
  - rearrange I/O requests to allow more efficient use of the I/O device
    - disk scheduling
  - Use buffering & caching

Error handling

Protection of the I/O devices

Kernel I/O Subsystem

- I/O Scheduling
  - **I/O request ordering via per-device queue**
  - **Example: disc scheduling**
  - **Goals: efficiency, fairness, priorities**
- Buffering **- store data in memory while transferring between devices**
  - **Double buffering**
  - **To cope with device speed mismatch**
  - **To cope with device transfer size mismatch**
  - **To maintain "copy semantics"**

Kernel I/O Subsystem

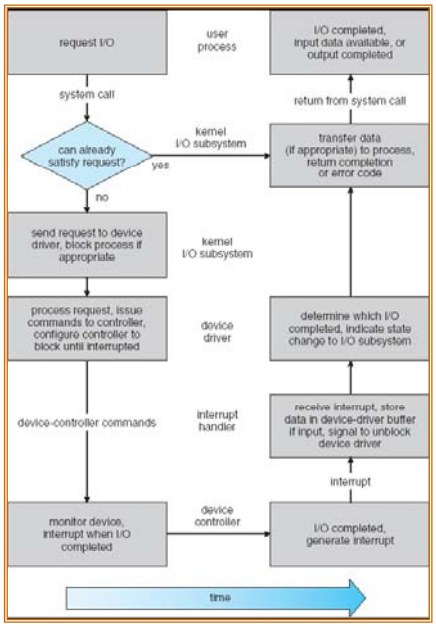
- Caching **- fast memory holding copy of data**
  - **Always just a copy**
  - **Key to performance**
- Spooling **- hold output for a device**
  - **If device can serve only one request at a time**
  - **i.e., Printing**
- Device reservation **- provides exclusive access to a device**
  - **System calls for allocation and deallocation**
  - **Watch out for deadlock**

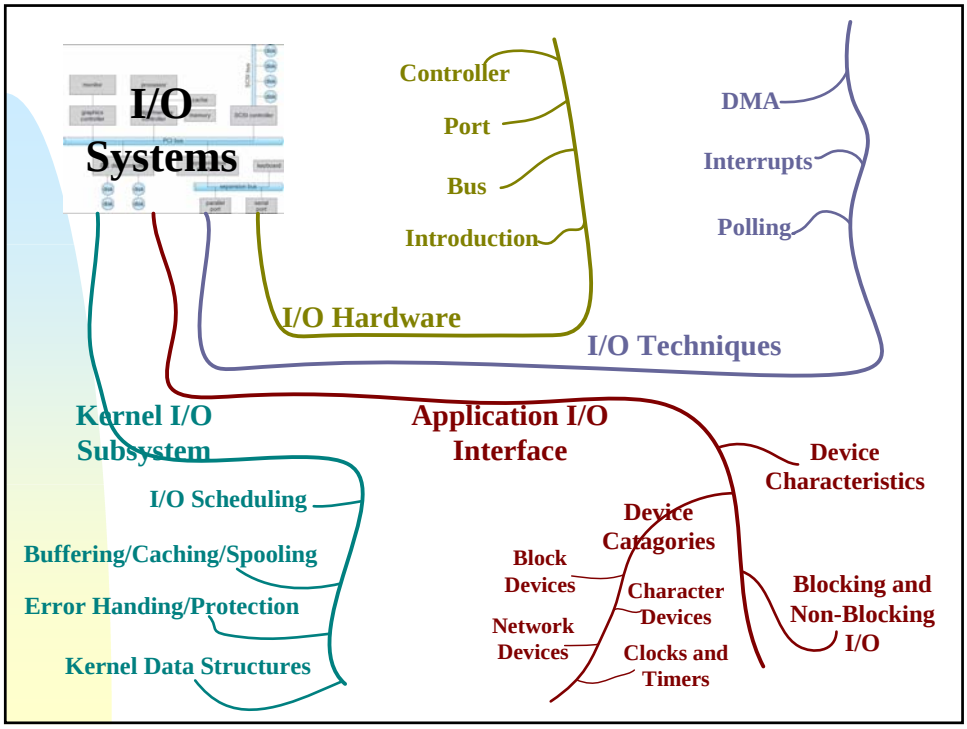
Kernel I/O Subsystem Summary

- Name space management – **which device is this file on?**
- Access control **– who can access this file/device?**
- Operations control **- which operations are possible on this device**
- File-System space allocation
- Device allocation
- Buffering, caching, spooling
- I/O scheduling
- Device-status monitoring, error handling and failure recovery
- Device driver configuration and initialization

Device drivers provide a uniform interface for accessing devices to the upper layers

Example: I/O Request Life Cycle




I/O Systems
Controller
Port
Bus
Introduction
I/O Hardware
DMA
Interrupts
Polling
I/O Techniques
Kernel I/O Subsystem
I/O Scheduling
Buffering/Caching/Spooling
Error Handing/Protection
Kernel Data Structures
Application I/O Interface
Device Characteristics
Device Catagories
Block Devices
Character Devices
Network Devices
Clocks and Timers
Blocking and Non-Blocking I/O