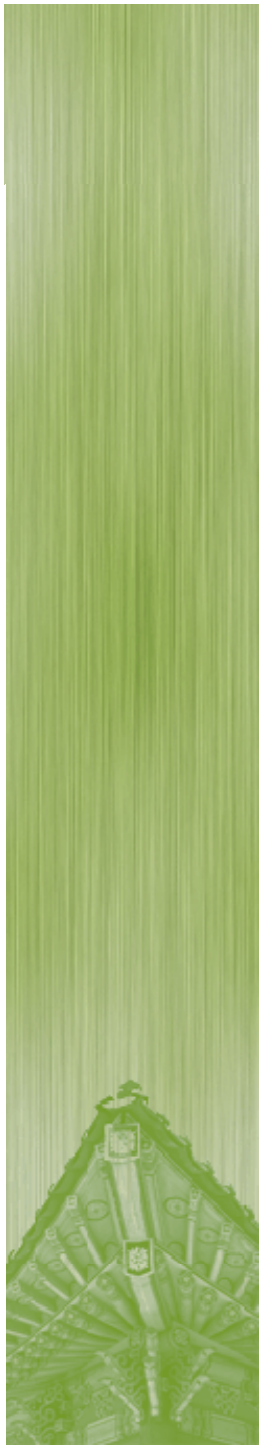


I/O Systems

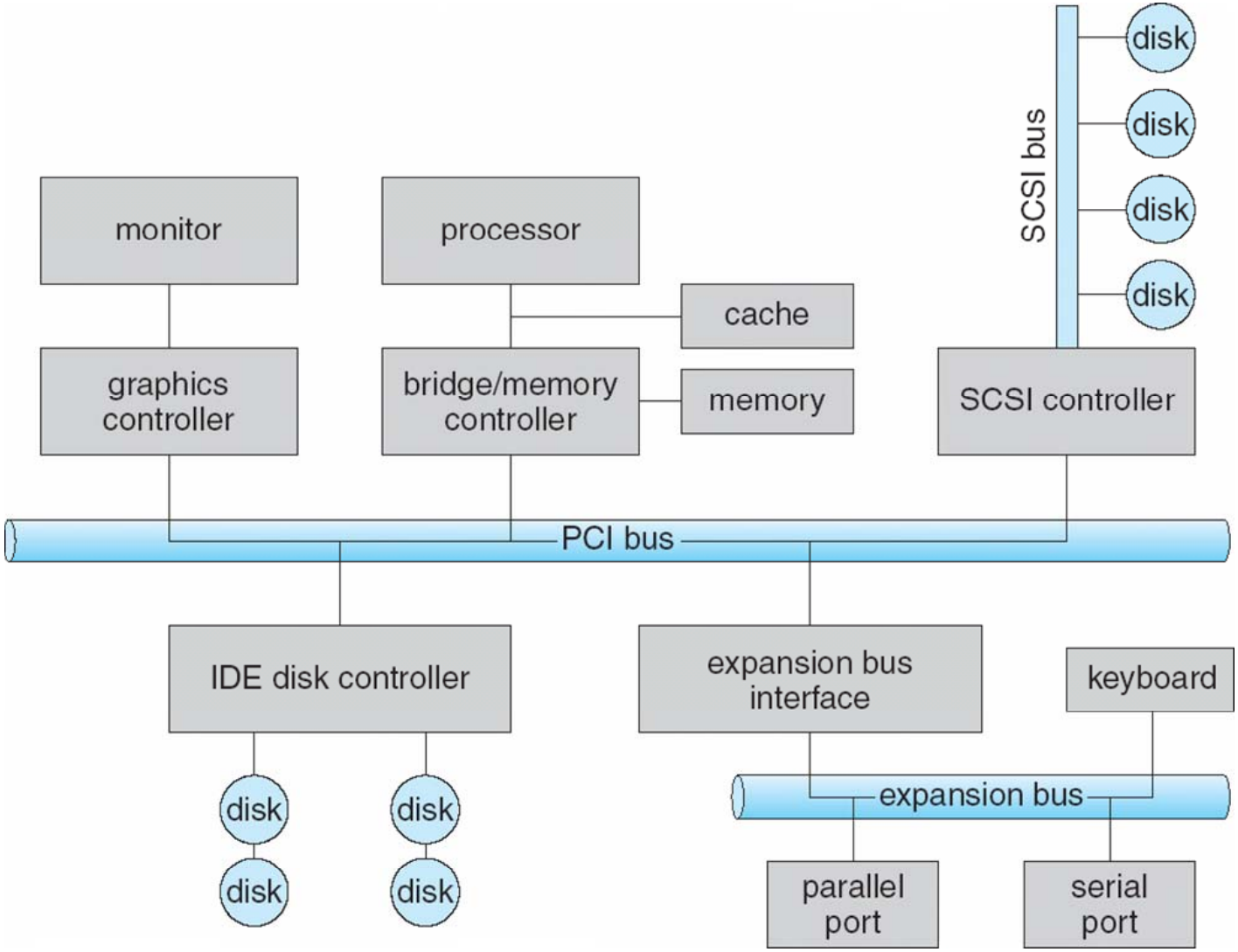
Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Today's Topics

- **Device characteristics**
  - Block device vs. Character device
  - Direct I/O vs. Memory-mapped I/O
  - Polling vs. Interrupts
  - Programmed I/O vs. DMA
  - Blocking vs. Non-blocking I/O

- **I/O software layers**

A Typical PC Bus Structure



I/O Devices (1)

- **Block device**
  - Stores information in fixed-size blocks, each one with its own address.
  - 512B – 32KB per block
  - It is possible to read or write each block independently of all the other ones.
  - Disks, tapes, etc.
- **Character device**
  - Delivers or accepts a stream of characters.
  - Not addressable and no seek operation.
  - Printers, networks, mice, keyboards, etc.

I/O Devices (2)

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

USB 2.0: 60 MB/s

I/O Devices (3)

- **Device controller (or host adapter)**
  - I/O devices have components:
    - Mechanical component
    - Electronic component
  - The electronic component is the device controller.
    - May be able to handle multiple devices.
  - Controller's tasks
    - Convert serial bit stream to block of bytes.
    - Perform error correction as necessary.
    - Make available to main memory.

Accessing I/O Devices (1)

- **Direct I/O**
  - Use special I/O instructions to an I/O port address.

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)

Accessing I/O Devices (2)

- **Memory-mapped I/O**
  - The device control registers are mapped into the address space of the processor.
    - The CPU executes I/O requests using the standard data transfer instructions.
  - I/O device drivers can be written entirely in C.
  - No special protection mechanism is needed to keep user processes from performing I/O
    - Can give a user control over specific devices but not others by simply including the desired pages in its page table.
  - Reading a device register and testing its value is done with a single instruction.

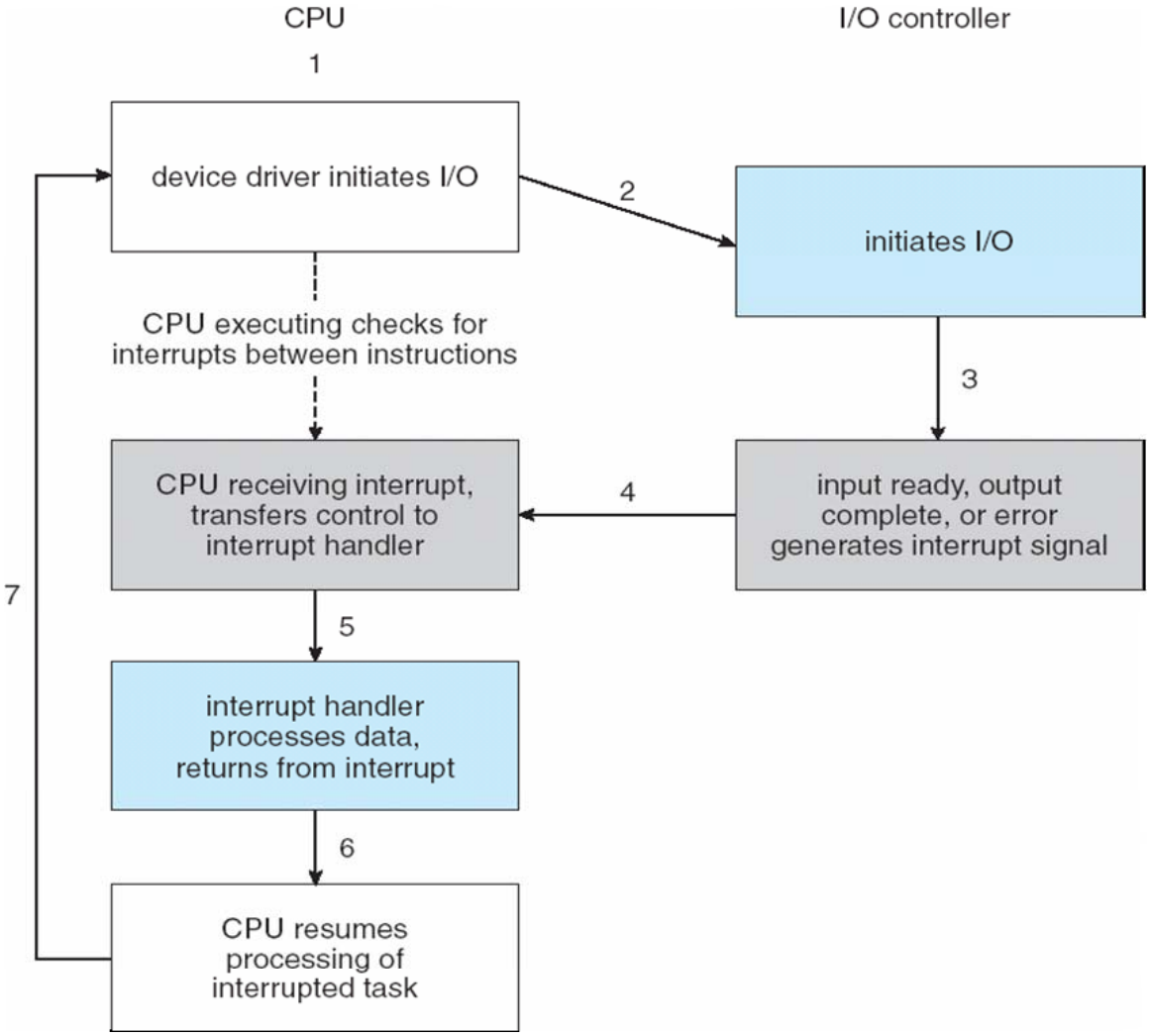
Polling vs. Interrupts (1)

- **Polled I/O**
  - CPU asks ("polls") devices if need attention.
    - ready to receive a command
    - command status, etc.
  - Advantages
    - Simple
    - Software is in control.
    - Efficient if CPU finds a device to be ready soon.
  - Disadvantages
    - Inefficient in non-trivial system (high CPU utilization).
    - Low priority devices may never be serviced.

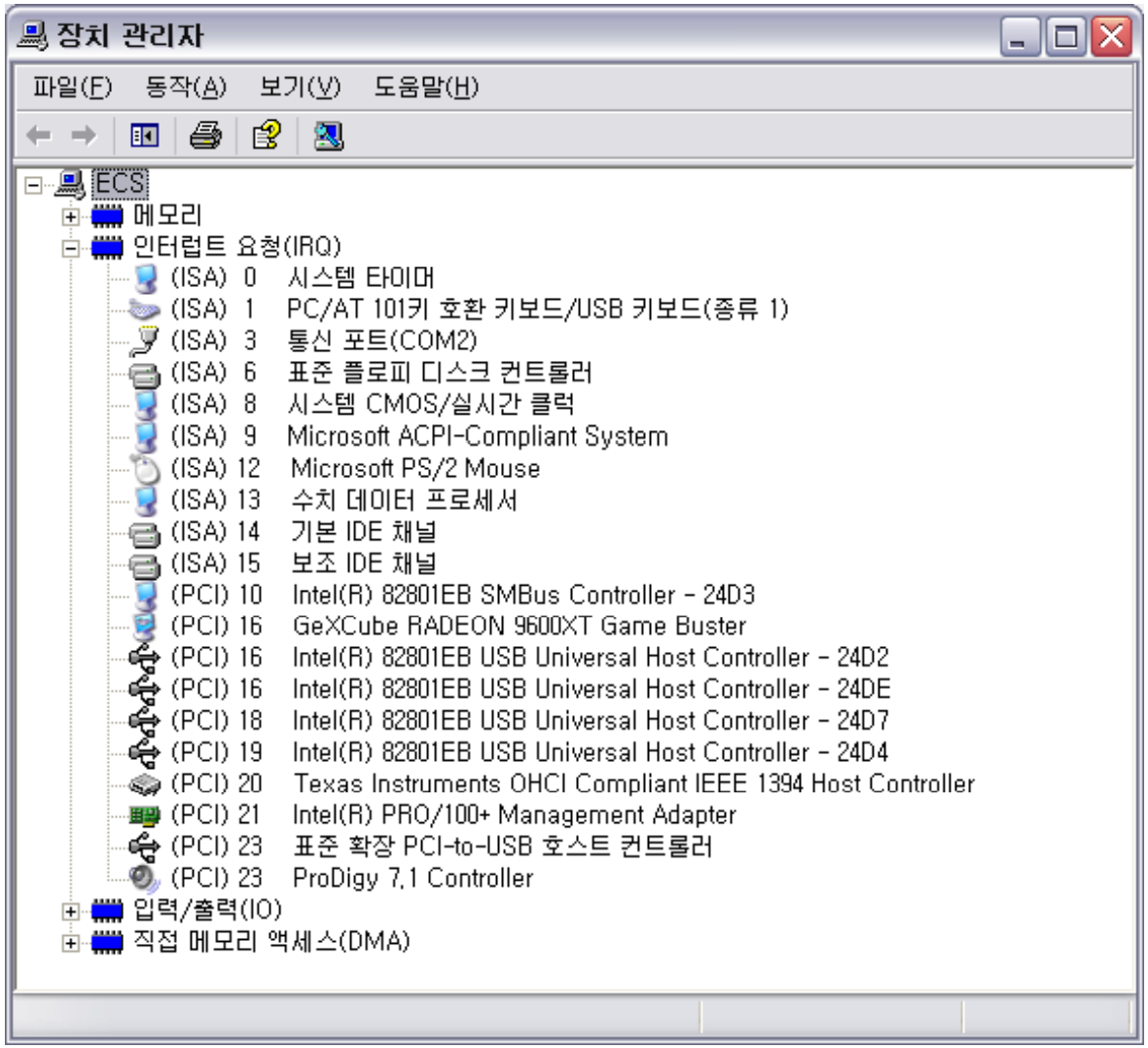
Polling vs. Interrupts (2)

- **Interrupt-driven I/O**
  - I/O devices request interrupt when need attention.
  - Interrupt service routines specific to each device are invoked.
  - Interrupts can be shared between multiple devices.
  - Advantages
    - CPU only attends to device when necessary.
    - More efficient than polling in general.
  - Disadvantages
    - Excess interrupts slow (or prevent) program execution.
    - Overheads (may need 1 interrupt per byte transferred)

Polling vs. Interrupts (3)

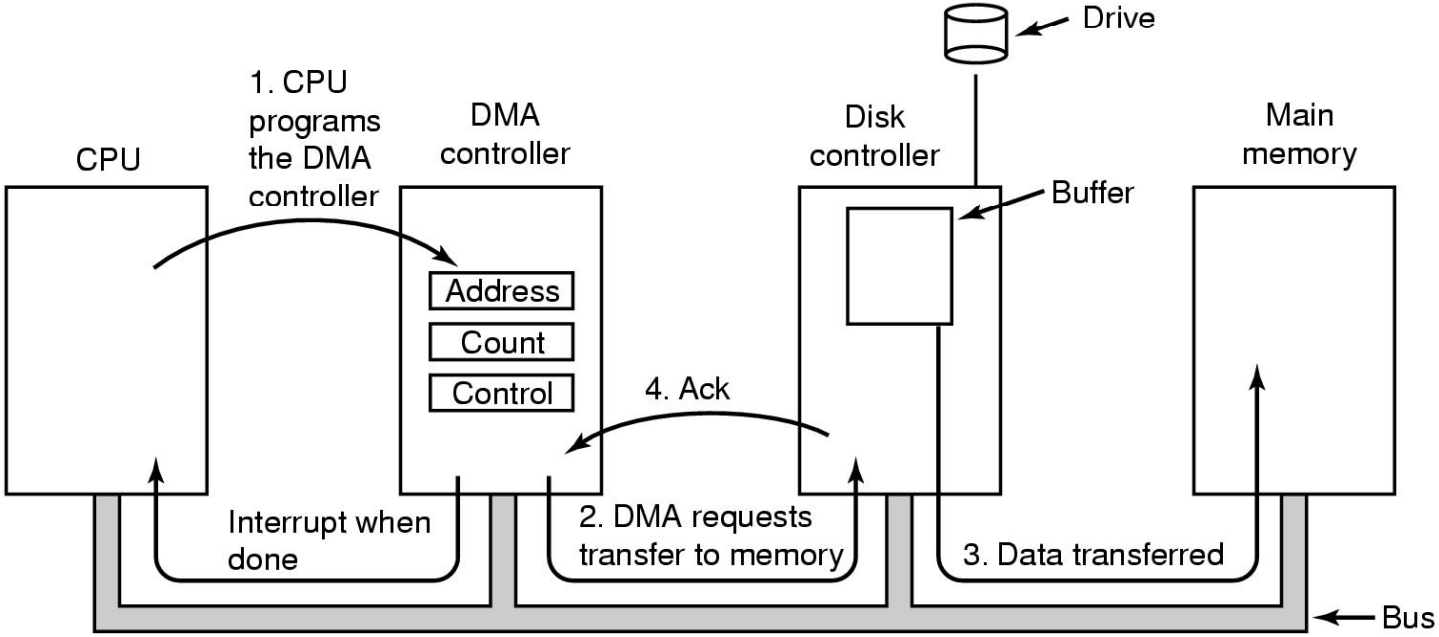


Polling vs. Interrupts (4)



Programmed I/O vs. DMA

- **DMA (Direct Memory Access)**
  - Bypasses CPU to transfer data directly between I/O device and memory.
  - Used to avoid programmed I/O for large data movement.



Blocking vs. Non-Blocking I/O

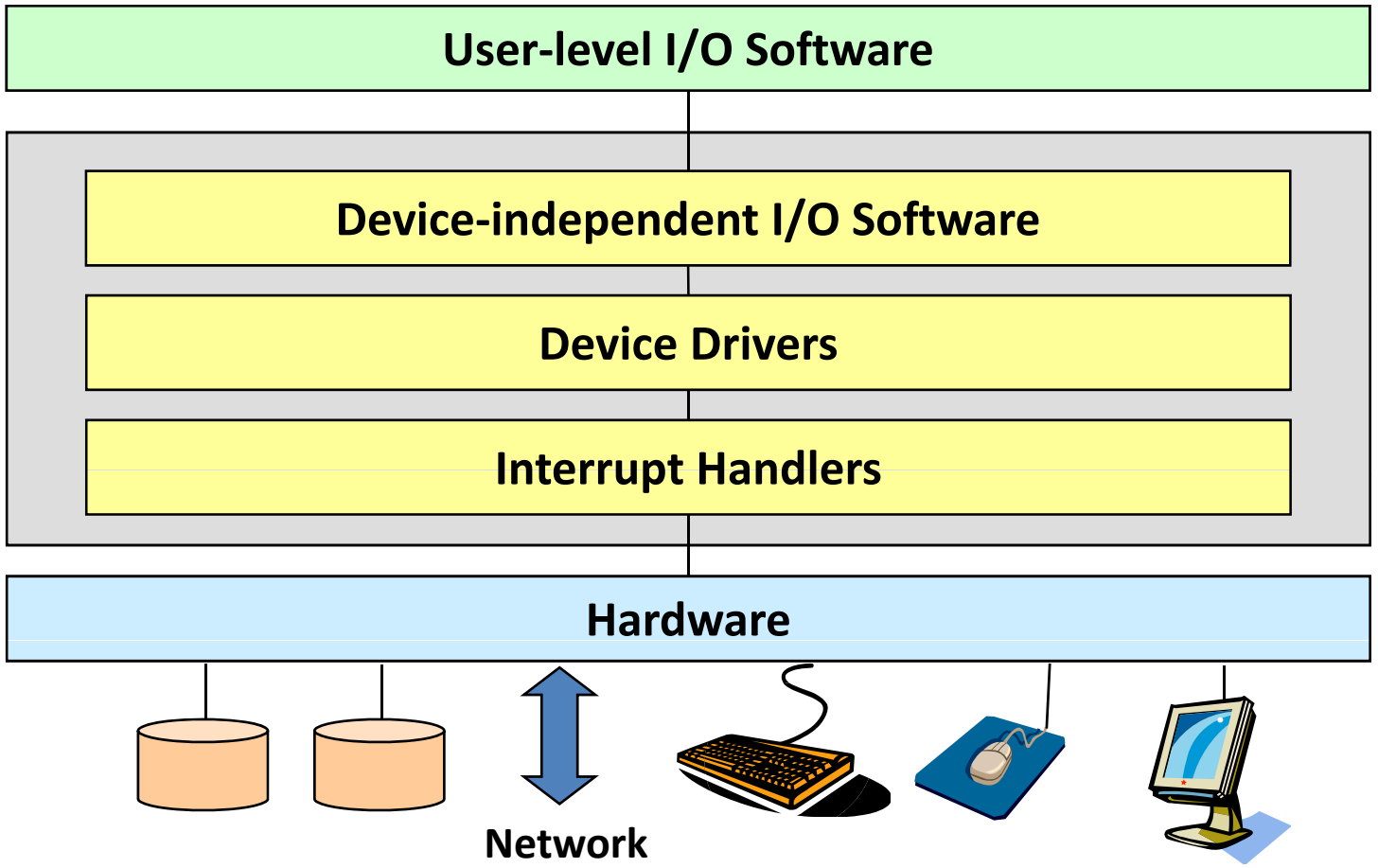
- **Blocking I/O**
  - Process is suspended until I/O completed.
  - Easy to use and understand.

- **Nonblocking I/O**
  - I/O call returns quickly, with a return value that indicates how many bytes were transferred.
  - A nonblocking `read()` returns immediately with whatever data available – the full number of bytes requested, fewer, or none at all.

Goals of I/O Software

- **Goals**
  - Device independence
    - Programs can access any I/O device without specifying device in advance.
  - Uniform naming
    - Name of a file or device should simply be a string or an integer.
  - Error handling
    - Handle as close to the hardware as possible.
  - Synchronous vs. asynchronous
    - blocked transfers vs. interrupt-driven
  - Buffering
    - Data coming off a device cannot be stored in final destination.
  - Sharable vs. dedicated devices
    - Disks vs. tape drives
    - Unsharable devices introduce problems such as deadlocks.

I/O Software Layers



Interrupt Handlers

- **Handling interrupts**

Critical actions

: Acknowledge an interrupt to the PIC.
: Reprogram the PIC or the device controller.
: Update data structures accessed by both the device and the processor.

Reenable interrupts

Noncritical actions

: Update data structures that are accessed only by the processor.
(e.g., reading the scan code from the keyboard)

Return from interrupts

Noncritical deferred actions

: Actions may be delayed.
: Copy buffer contents into the address space of some process (e.g., sending the keyboard line buffer to the terminal handler process).
: Bottom half (Linux)

Device Drivers (1)

- **Device drivers**
  - Device-specific code to control each I/O device interacting with device-independent I/O software and interrupt handlers.
  - Requires to define a well-defined model and a standard interface of how they interact with the rest of the OS.
  - Implementing device drivers:
    - Statically linked with the kernel.
    - Selectively loaded into the system during boot time.
    - Dynamically loaded into the system during execution. (especially for hot pluggable devices).

Device Drivers (2)

Device Drivers (3)

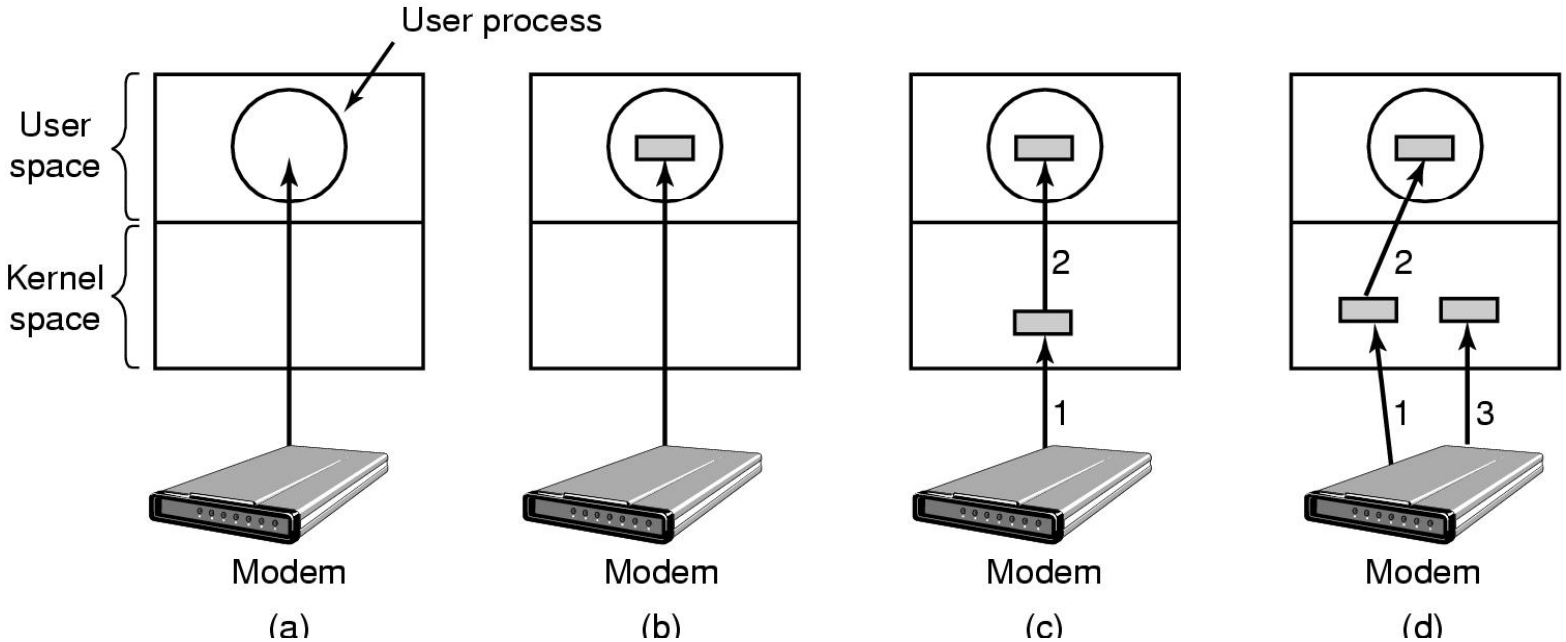
- **The problem**
  - Reliability remains a crucial, but unresolved problem
    - 5% of Windows systems crash every day
    - Huge cost of failures: stock exchange, e-commerce, …
    - Growing "unmanaged systems": digital appliances, consumer electronics devices
  - OS extensions are increasingly prevalent
    - 70% of Linux kernel code
    - Over 35,000 drivers with over 120,000 versions on Windows XP
    - Written by less experienced programmer
  - Extensions are a leading cause of OS failure
    - Drivers cause 85% of Windows XP crashes
    - Drivers are 7 times buggier than the kernel in Linux

Device-Independent I/O SW (1)

- **Uniform interfacing for device drivers**
  - In Unix, devices are modeled as special files.
    - They are accessed through the use of system calls such as open(), read(), write(), close(), ioctl(), etc.
    - A file name is associated with each device.
  - Major device number locates the appropriate driver.
    - Minor device number (stored in i-node) is passed as a parameter to the driver in order to specify the unit to be read or written.
  - The usual protection rules for files also apply to I/O devices.

Device-Independent I/O SW (2)

- **Buffering**
  - (a) Unbuffered
  - (b) Buffered in user space
  - (c) Buffered in the kernel space
  - (d) Double buffering in the kernel



Device-Independent I/O SW (3)

- **Error reporting**
  - Many errors are device-specific and must be handled by the appropriate driver, but the framework for error handling is device independent.
  - Programming errors vs. actual I/O errors
  - Handling errors
    - Returning the system call with an error code.
    - Retrying a certain number of times.
    - Ignoring the error.
    - Killing the calling process.
    - Terminating the system.

Device-Independent I/O SW (4)

- **Allocating and releasing dedicated devices**
  - Some devices cannot be shared.

  (1) Require processes to perform open()'s on the special files for devices directly.
    - The process retries if open() fails.

  (2) Have special mechanisms for requesting and releasing dedicated devices.
    - An attempt to acquire a device that is not available blocks the caller.
- **Device-independent block size**
  - Treat several sectors as a single logical block.
  - The higher layers only deal with abstract devices that all use the same block size.

User-Space I/O Software

- **Provided as a library**
  - Standard I/O library in C
    - `fopen()` vs. `open()`?
    - Buffering for `fgetc()`?

- **Spooling**
  - A way of dealing with dedicated I/O devices in a multiprogramming system.
  - Implemented by a daemon and a spooling directory.
  - Printers, network file transfers, USENET news, mails, etc.

I/O Systems Layers