I/O Systems

Jo, Heeseung

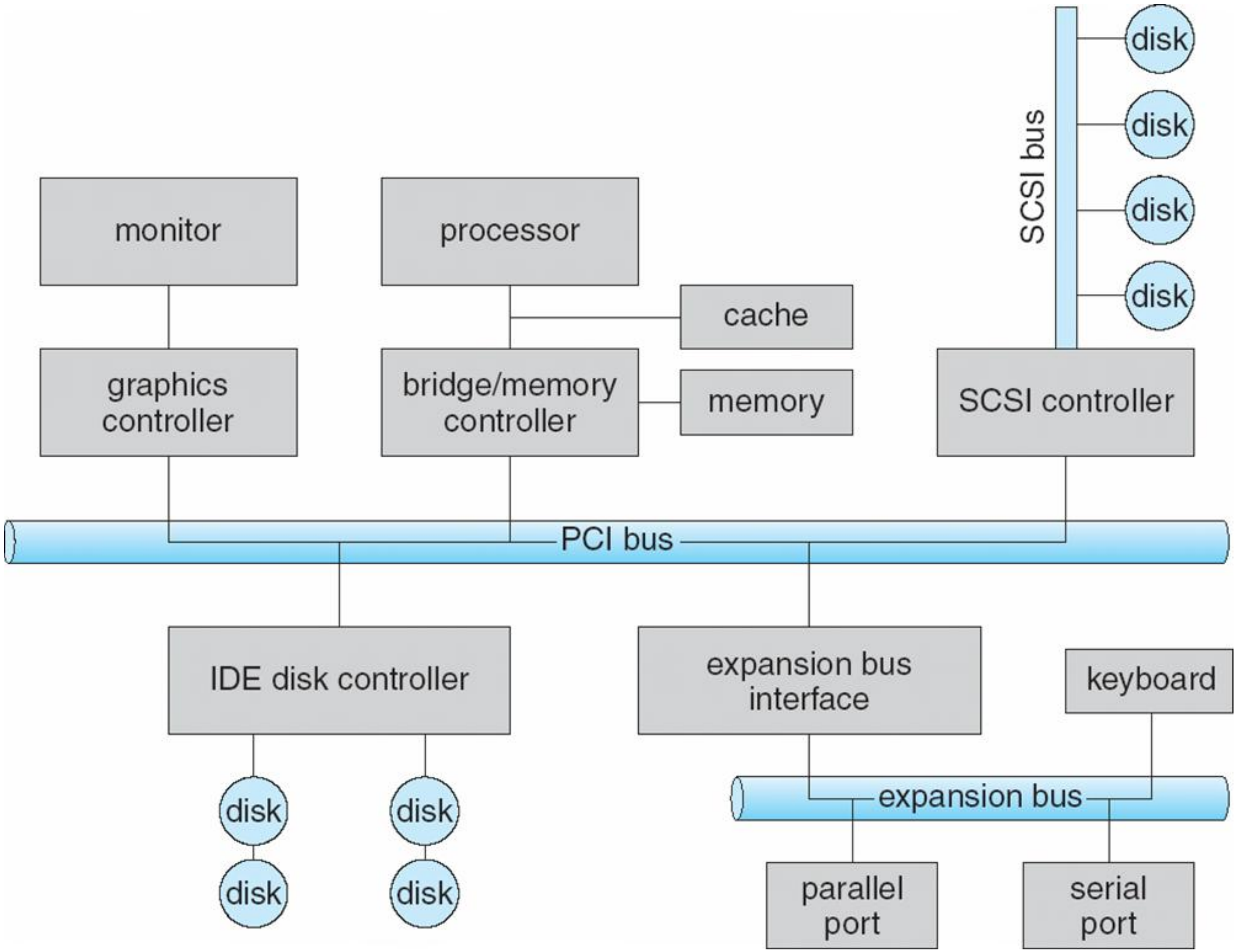
Today's Topics

Device characteristics

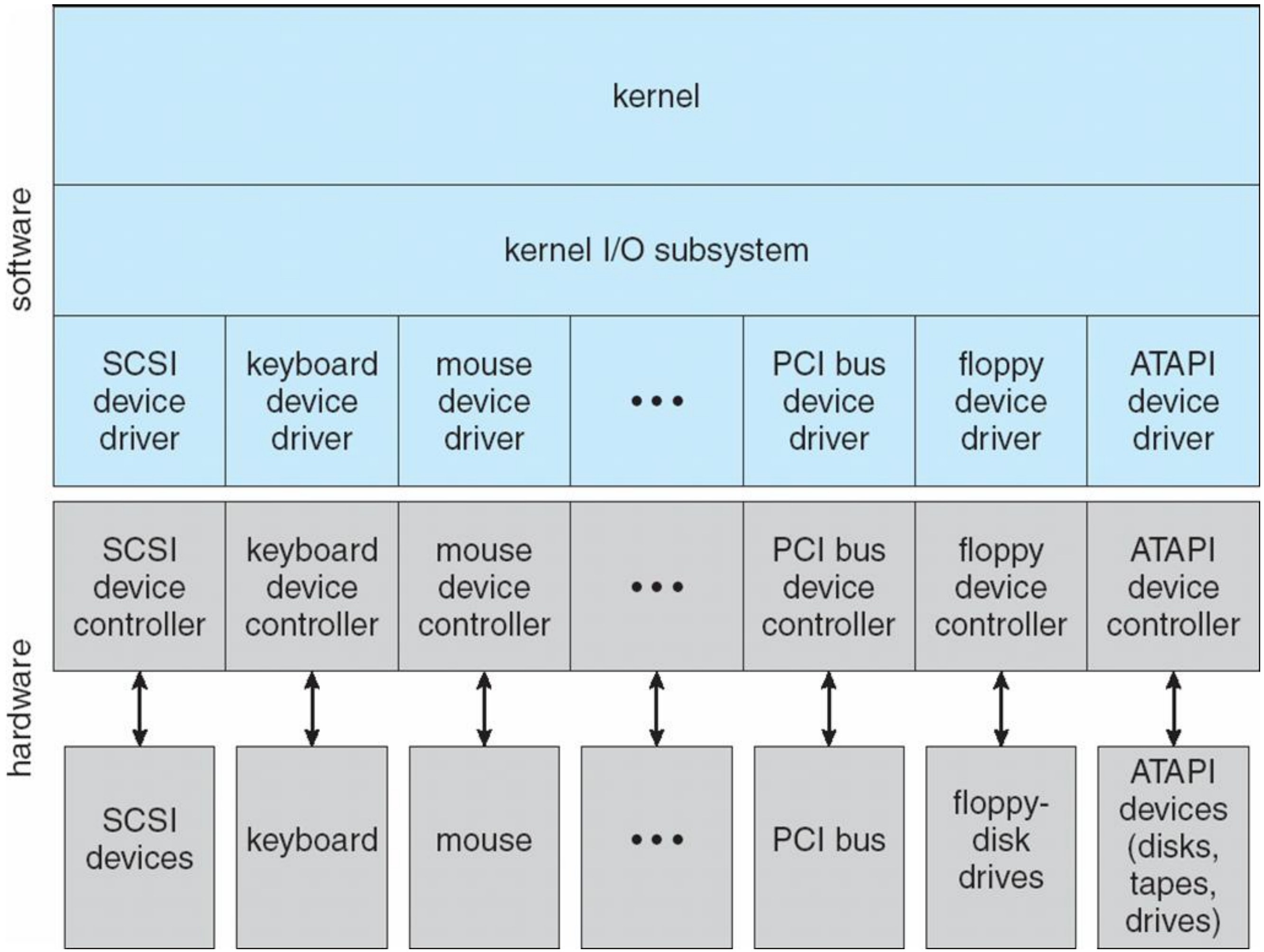
- Block device vs. Character device
- Direct I/O vs. Memory-mapped I/O
- Polling vs. Interrupts
- Programmed I/O vs. DMA
- Blocking vs. Non-blocking I/O

I/O software layers

A Typical PC Bus Structure



A Kernel I/O Structure



I/O Devices (1)

Block device

- Stores information in fixed-size blocks
- Each one with its own address
- 512B – 32KB per block
- 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, modem, mice, keyboards, etc.

I/O Devices (2)

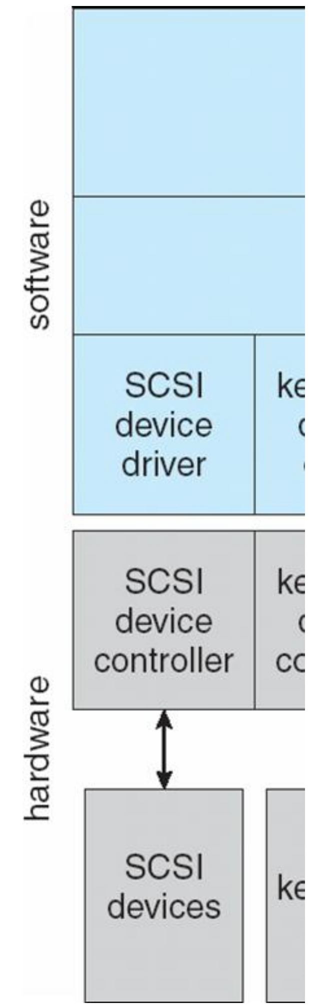
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
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
USB 3.0:   625 MB/s

SATA:      1.5 Gb/s
SATA2:       3 Gb/s
SATA3:       6 Gb/s
SATA M.2:    6 Gb/s
NVMe M.2:   20 Gb/s

PCIe 1.0: 250 MB/s
PCIe 2.0: 500 MB/s
PCIe 3.0: 984 MB/s
PCIe 4.0: 1.9 GB/s
```

I/O Devices (3)

Device controller (or host adapter)

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



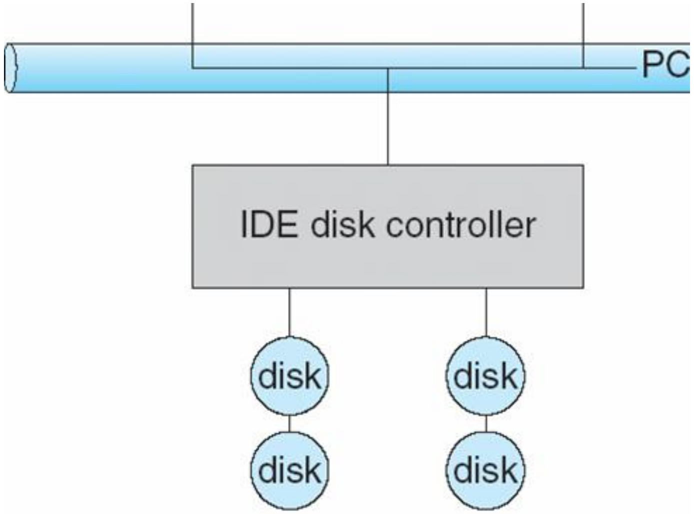
I/O Hardware

Devices usually have registers

- Device driver places commands, addresses, and data to write, or read data from registers after command execution
- Data-in register, data-out register, status register, control register
- Typically 1-4 bytes, or FIFO buffer

Devices have addresses, used by

- Direct I/O instructions
- Memory-mapped I/O
  - Device data and command registers mapped to processor address space



Accessing I/O Devices (1)

Direct I/O

- Use special I/O instructions to an I/O port address
- e.g.) Intel I/O instructions
  - in : Reads from a port
  - ins: Inputs a string from a port
  - insb: Inputs a byte string from a port
  - insl: Inputs a doubleword string from a port
  - insw: Inputs a word string
  - out: Writes to a port

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 (load / store)
- I/O device drivers can be written entirely in C
- No special protection mechanism is needed
  - Protection via PTE
  - Including the desired pages in its page table
  - Can give a user control over specific devices
- Reading a device register and testing its value is done with a single instruction

Polling vs. Interrupts (1)

How can a CPU knows I/O requests are processed?

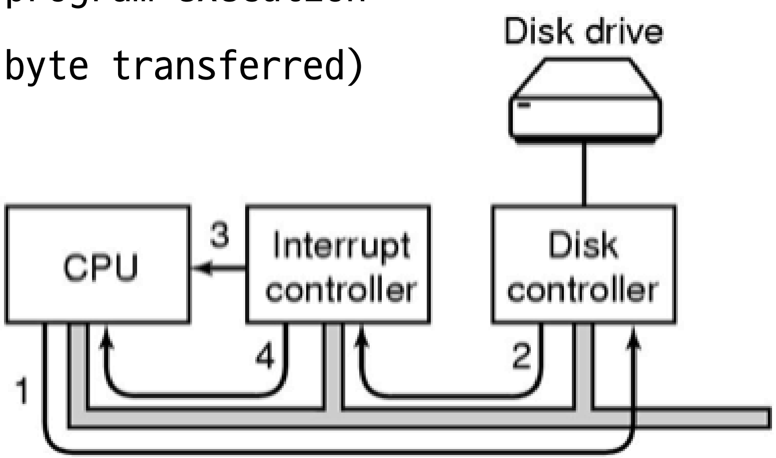
Polled I/O

- 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)

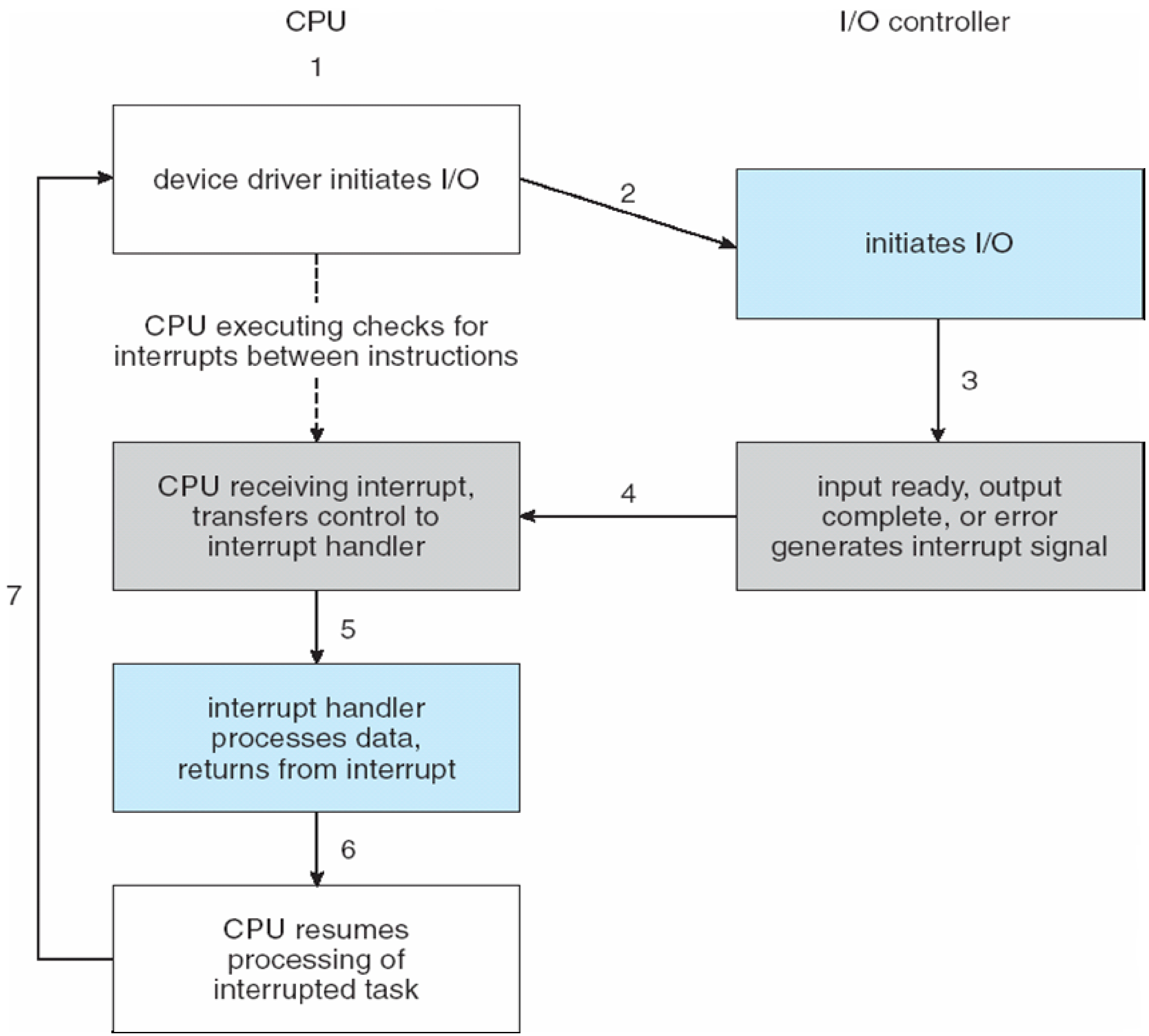
Interrupt-driven I/O

- 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)

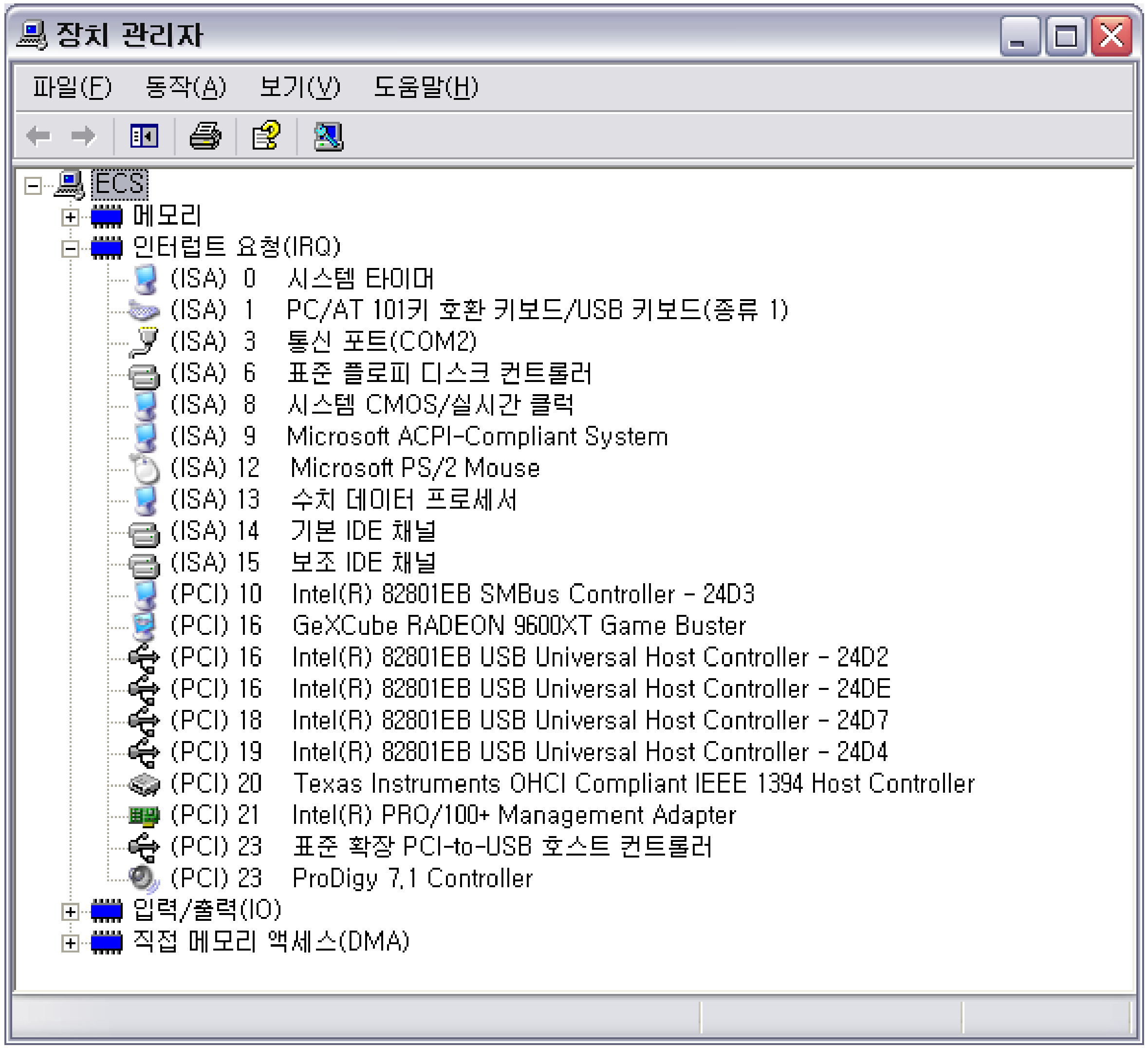


Polling vs. Interrupts (3)

Interrupt-driven I/O



Polling vs. Interrupts (4)



Programmed I/O vs. DMA (1)

Data transfer modes in I/O

Programmed I/O (PIO)

- CPU is involved in moving data between I/O devices and memory
- By special I/O instructions vs. by memory-mapped I/O

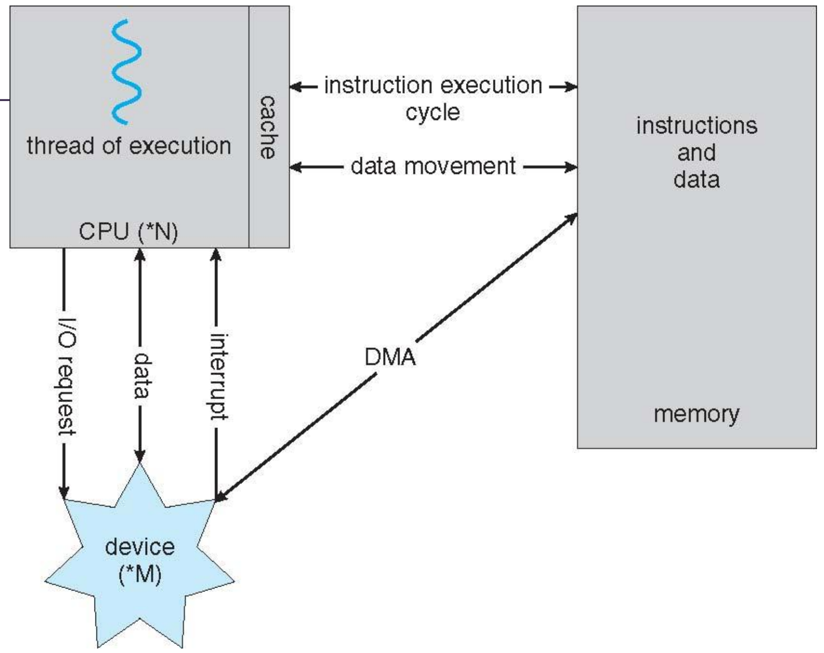
DMA (Direct Memory Access)

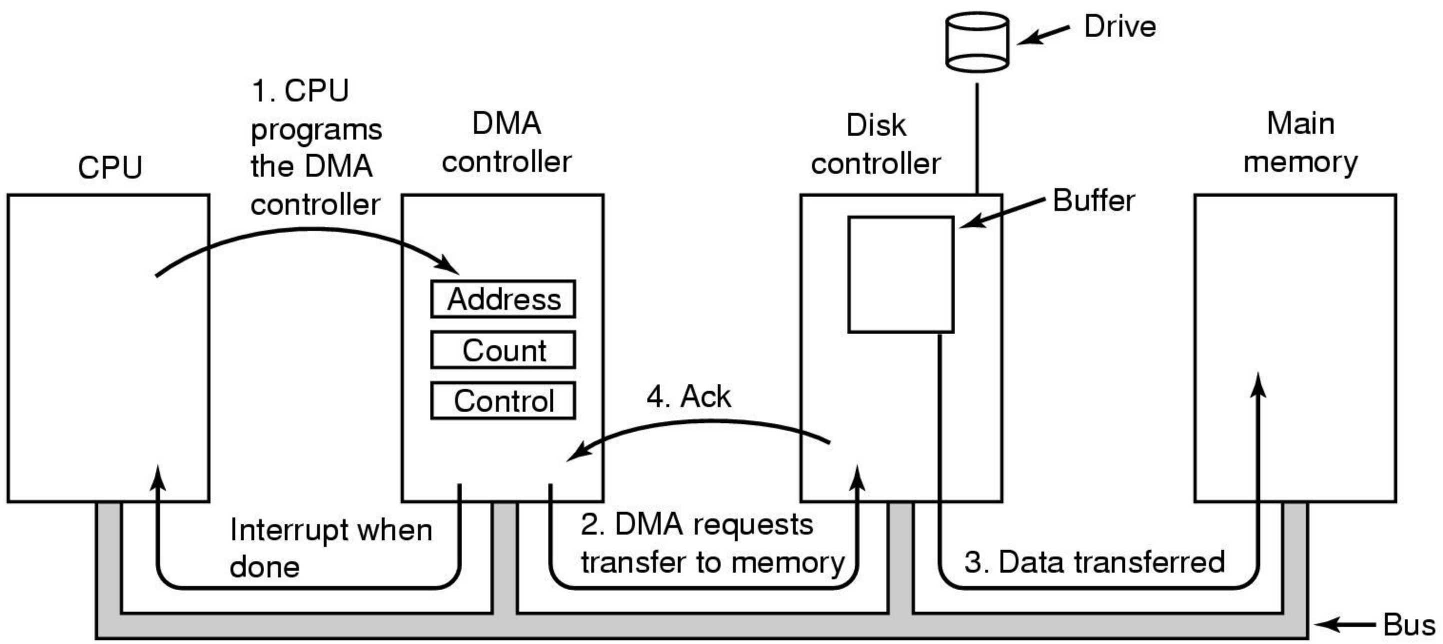
- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory
  - Without CPU intervention
- Only an interrupt is generated per block

Programmed I/O vs. DMA (2)

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
- read(), write()

Non-blocking I/O

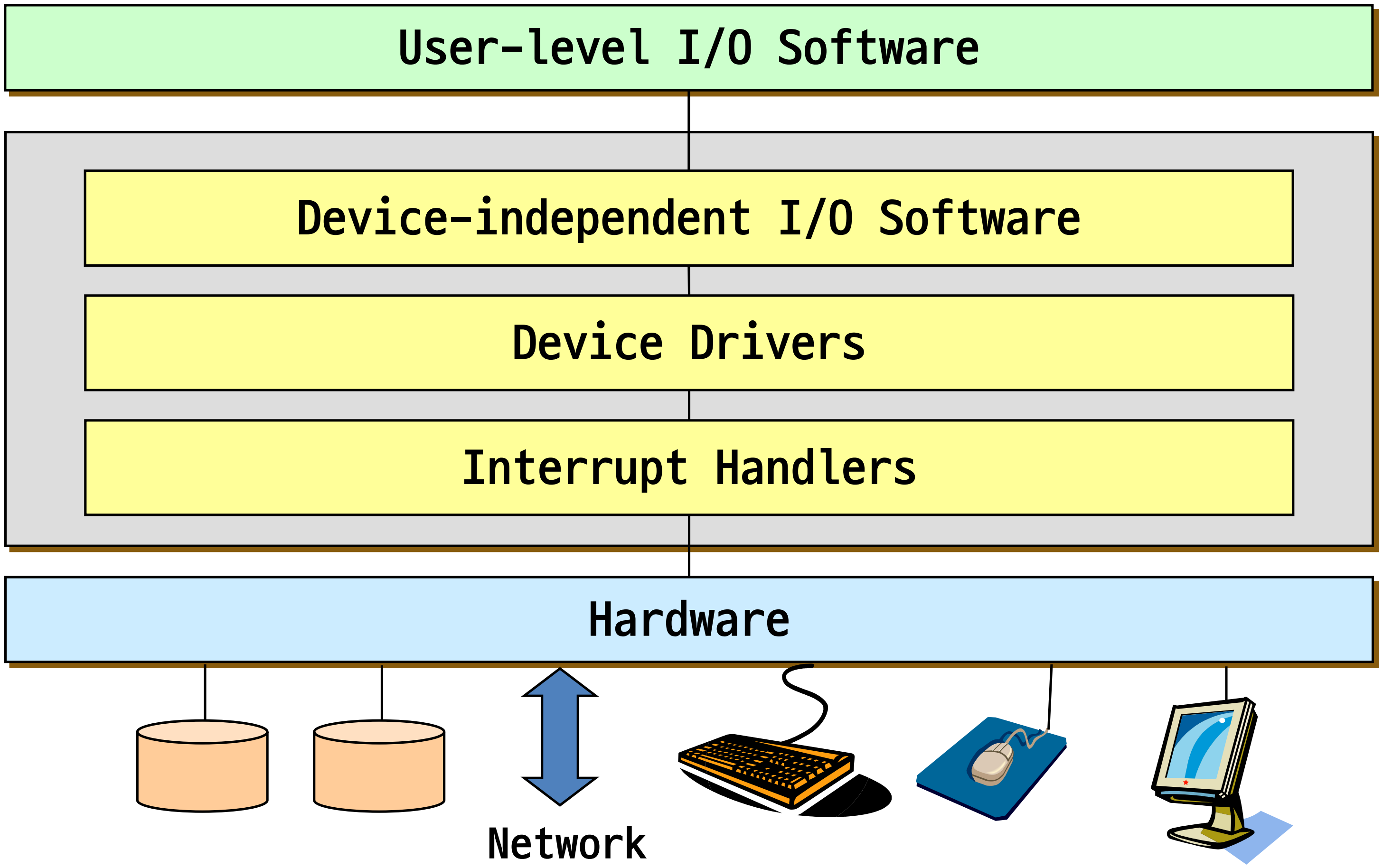
- I/O call returns quickly, with a return value that indicates how many bytes were transferred
- Implemented via multi-threading
- select() to find if data is ready

Goals of I/O Software

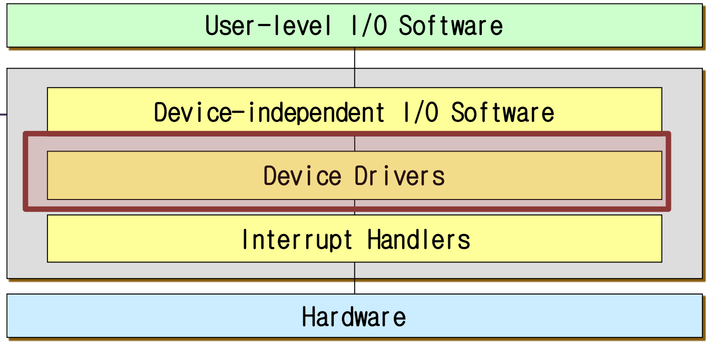
Goals

- Device independence
- Uniform naming
- Error handling
- Synchronous vs. asynchronous
- Buffering
- Sharable vs. dedicated devices

I/O Software Layers



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)

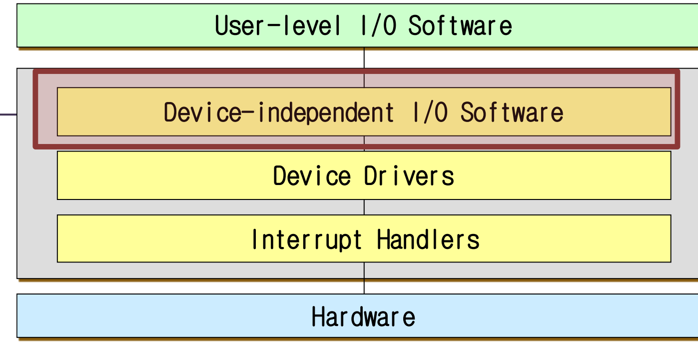
Device Drivers (2)

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
- Device driver 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
- 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
- The usual protection rules for files also apply to I/O devices
- /dev directory

```
crw-rw----  1 root lp     99,   0 2012-05-22 22:16 parport0
crw-rw----  1 root lp     99,   1 2012-05-22 22:16 parport1
crw-rw----  1 root lp     99,   2 2012-05-22 22:16 parport2
crw-rw----  1 root lp     99,   3 2012-05-22 22:16 parport3
crw-r-----  1 root kmem    1,   4 2012-05-22 22:16 port
crw-r--r--  1 root root   10, 135 2012-05-22 22:16 rtc
brw-r-----  1 root disk    8,   0 2012-05-22 22:16 sda
brw-r-----  1 root disk    8,   1 2012-05-22 22:16 sda1
brw-r-----  1 root disk    8,   2 2012-05-22 22:16 sda2
brw-r-----  1 root disk    8,   3 2012-05-22 22:16 sda3
```

Device-Independent I/O SW (2)

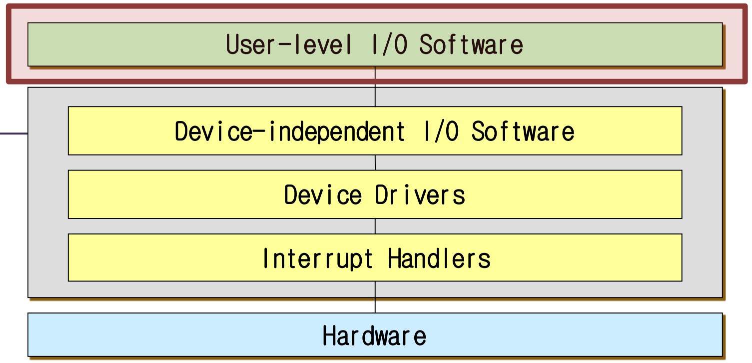
Error reporting

- Errors must be reported to user
- Many errors are device-specific
- Must be handled by the appropriate driver
- 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

User-Space I/O Software



Provided as a library

- Standard I/O library in C
  - fopen() vs. open()?

User's choice which library function to use

- fgets(), fscanf(), ...

Or, you can make your own I/O library

- myopen(), myfgets(), ...

I/O Systems Layers