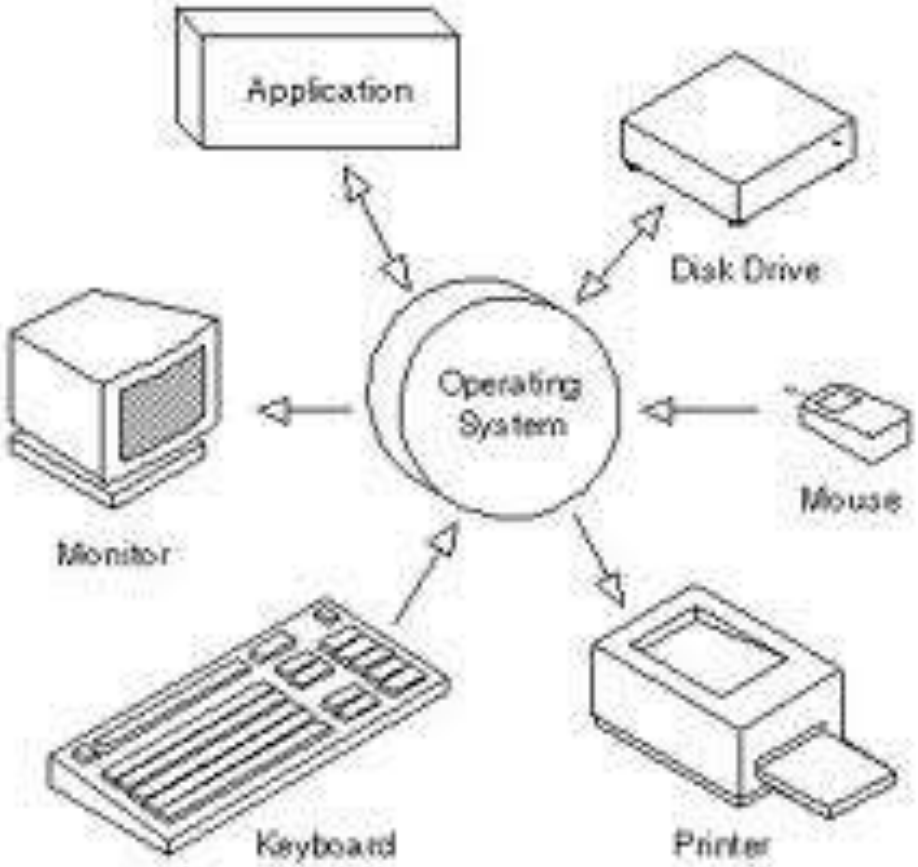
NYU
New York University

CSCI-GA.2250-001

Operating Systems

Lecture 9: I/O

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com


Application
Disk Drive
Operating System
Monitor
Mouse
Keyboard
Printer

Categories of I/O Devices

External devices that engage in I/O with computer systems can be grouped into three categories:

Human readable

- suitable for communicating with the computer user
- printers, terminals, video display, keyboard, mouse

Machine readable

- suitable for communicating with electronic equipment
- disk drives, USB keys, sensors, controllers

Communication

- suitable for communicating with remote devices
- modems, digital line drivers

A Simple Definition

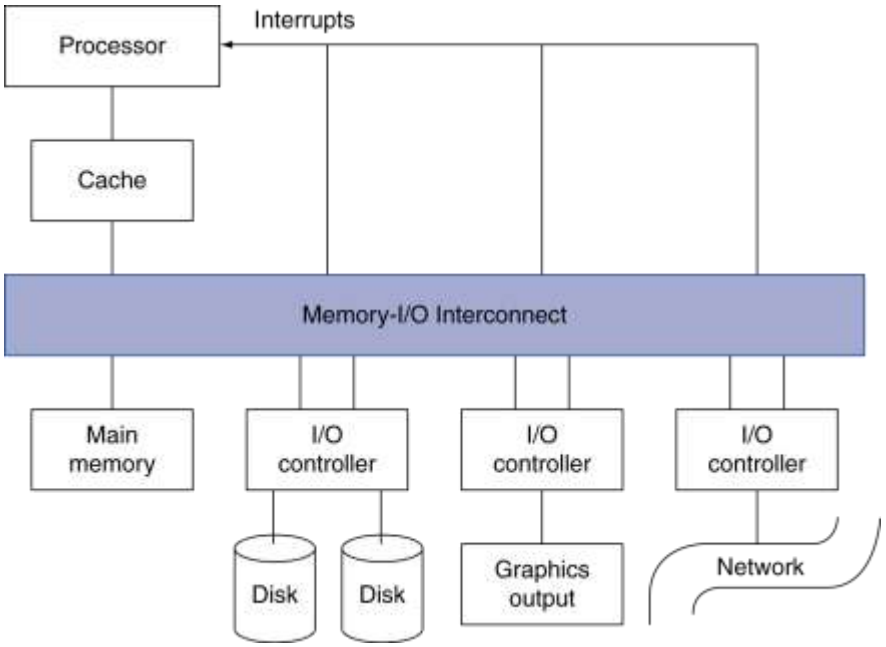
- The main concept of I/O is to move data from/to I/O devices to the processor using some modules and buffers.
- This is the way the processor deals with the outside world
- Examples: mouse, display, keyboard, disks, scanners, speakers, etc.

The OS and I/O

- The OS controls all I/O devices
  - Issue commands to devices
  - Catch interrupts
  - Handle errors
- Provides an interface between the devices and the rest of the system

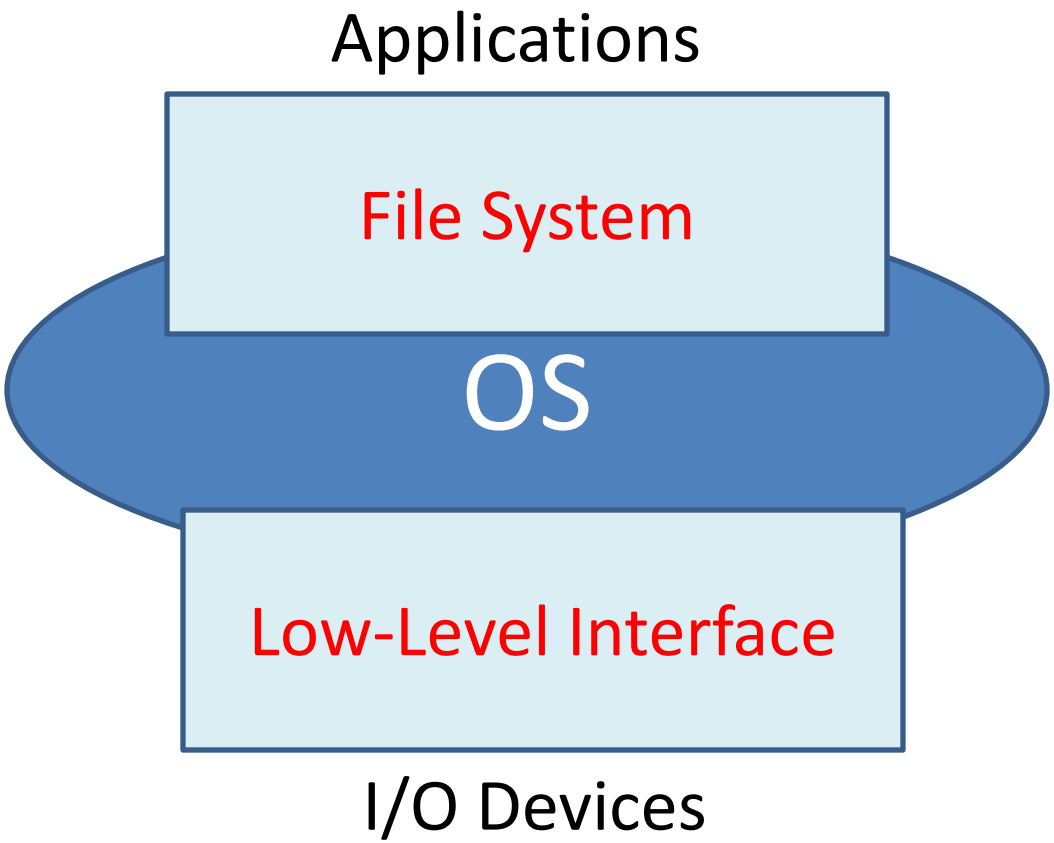
I/O Devices: Challenges

- Very diverse devices
  - behavior (i.e., input vs. output vs. storage)
  - partner (who is at the other end?)
  - data rate
- I/O Design affected by many factors (expandability, resilience)
- Performance:
  - access latency
  - throughput
  - connection between devices and the system
  - the memory hierarchy
  - the operating system
- A variety of different users

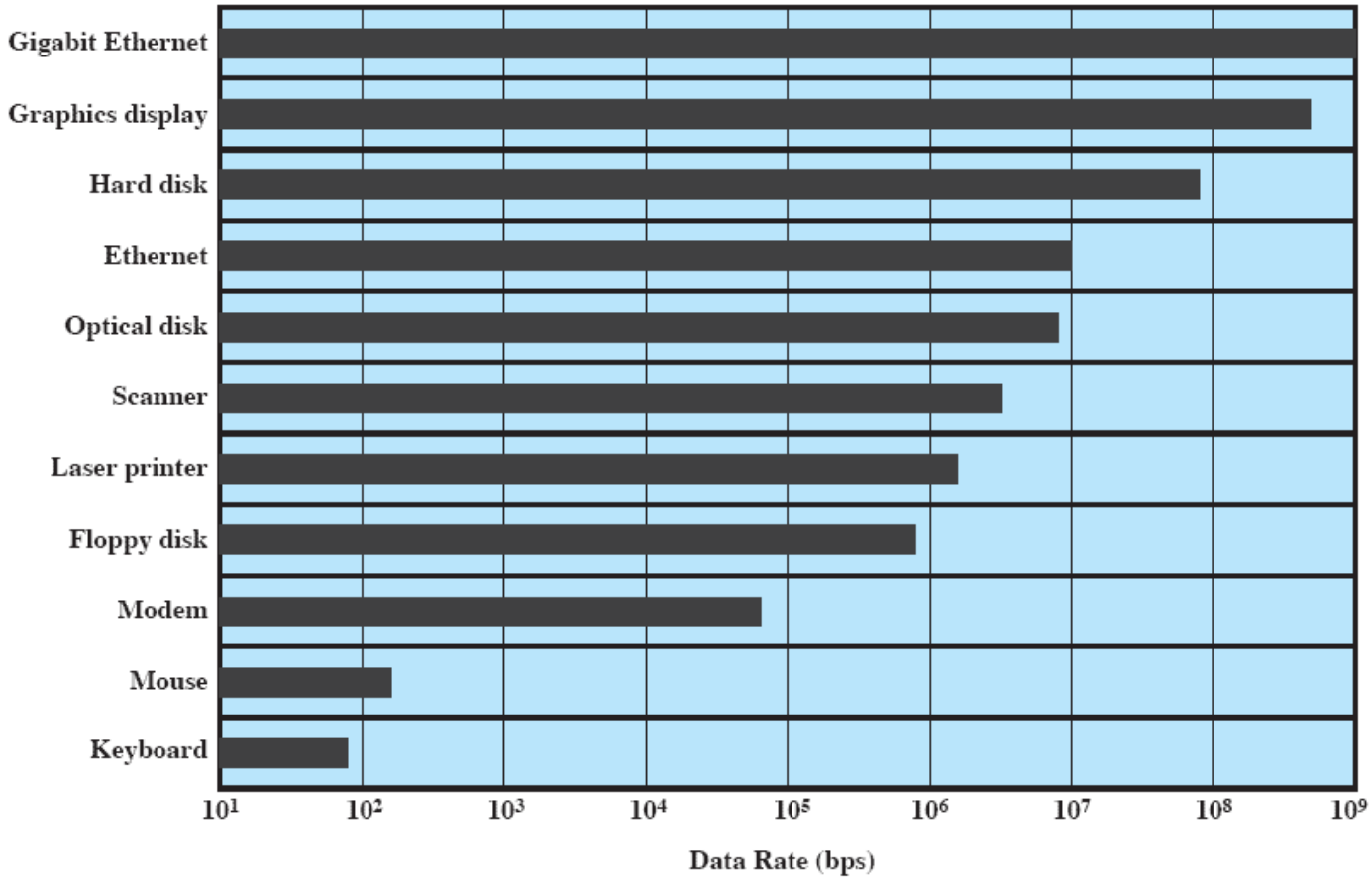


I/O Devices

- Block device
  - Stores information in fixed-size blocks
  - Each block has its own address
  - Transfers in one or more blocks
  - Example: Hard-disks, CD-ROMs, USB sticks
- Character device
  - Delivers or accepts stream of character
  - Is not addressable
  - Example: mice, printers, network interfaces


Applications
File System
OS
Low-Level Interface
I/O Devices

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	3.5 MB/sec
802.11g Wireless	6.75 MB/sec
52x CD-ROM	7.8 MB/sec
Fast Ethernet	12.5 MB/sec
Compact flash card	40 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
SATA disk drive	300 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec


Gigabit Ethernet
Graphics display
Hard disk
Ethernet
Optical disk
Scanner
Laser printer
Floppy disk
Modem
Mouse
Keyboard
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
Data Rate (bps)

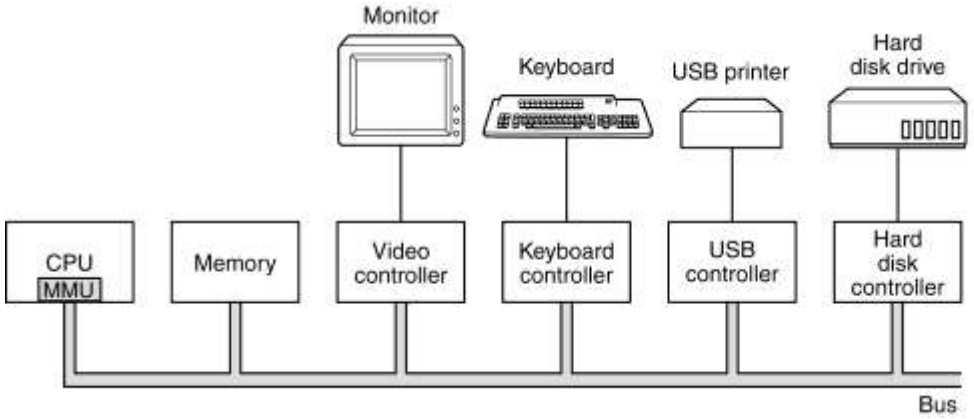
I/O Units

Mechanical component

Electronic Component

The Device Itself

Device Controller



Controller and Device

- Each controller has few registers used to communicate with CPU
- By writing/reading into/from those registers, the OS can control the devices.
- There are also data buffers in the device that can be read/written by the OS

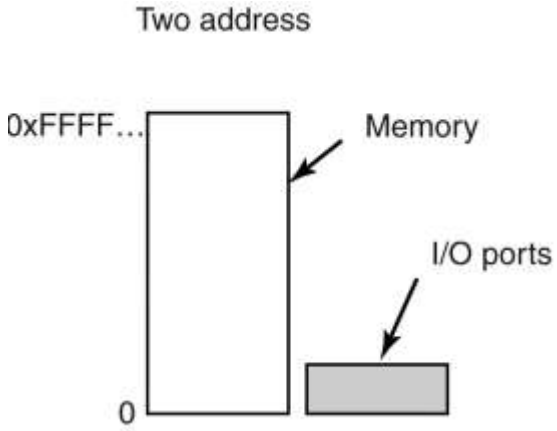
How does CPU communicate with control registers and data buffers?

Two main approaches

- I/O port space
- Memory-mapped I/O

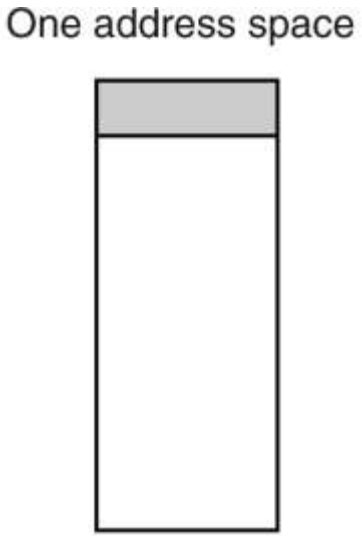
I/O Port Space

- Each control register is assigned an I/O port number
- The set of all I/O ports form the I/O port space
- I/O port space is protected



Memory-Mapped I/O

- Map control registers into the memory space
- Each control register is assigned a unique memory address

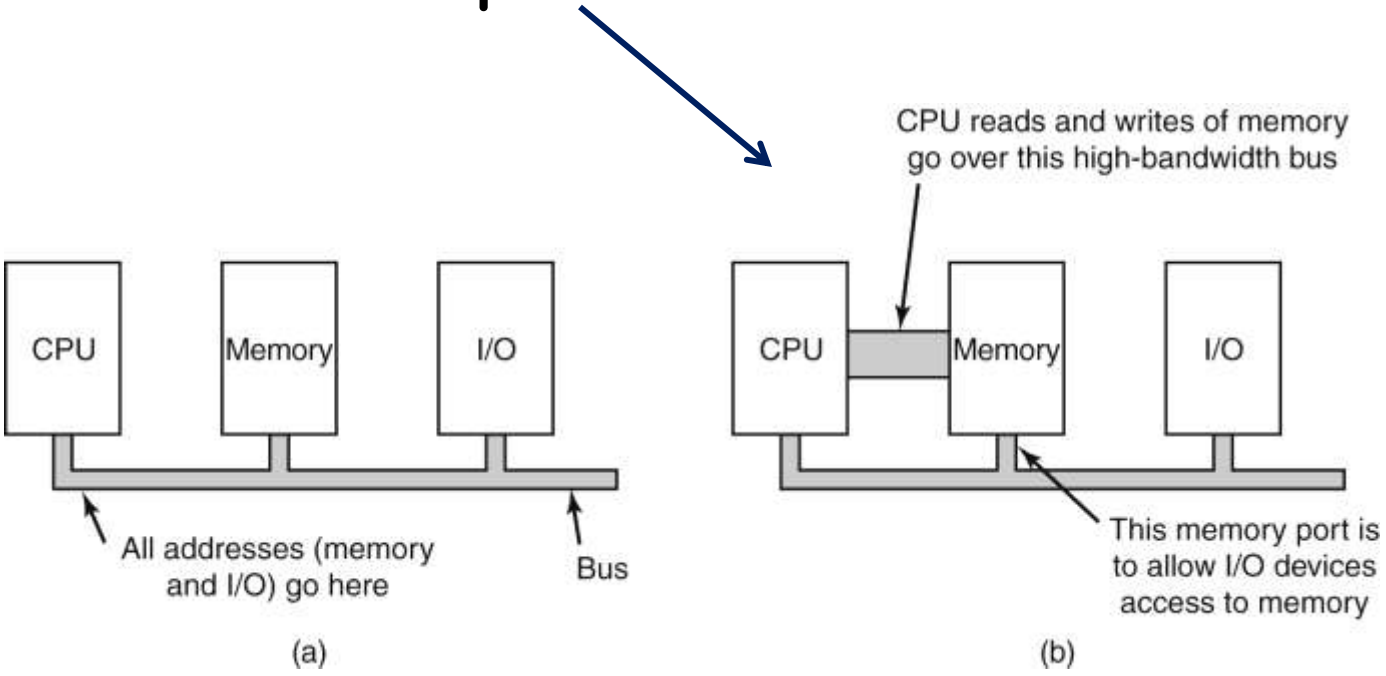


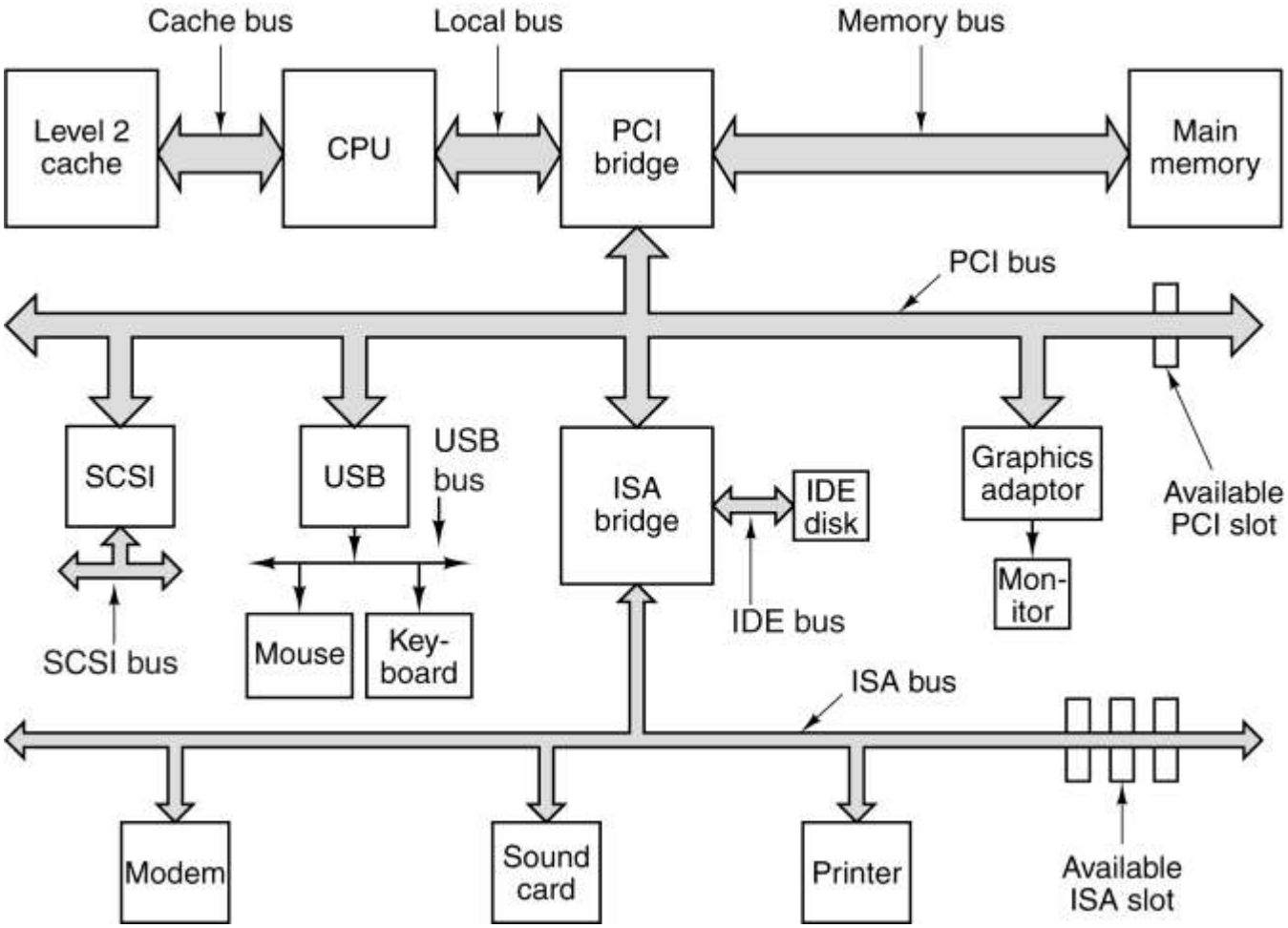
Advantages of Memory-Mapped I/O

- Device drivers can be written entirely in C (since no special instructions are needed)
- No special protection is needed from OS, just refrain from putting that portion of the address space in any user's virtual address space
- Every instruction that can reference memory can also reference control registers

Disadvantages of Memory-Mapped I/O

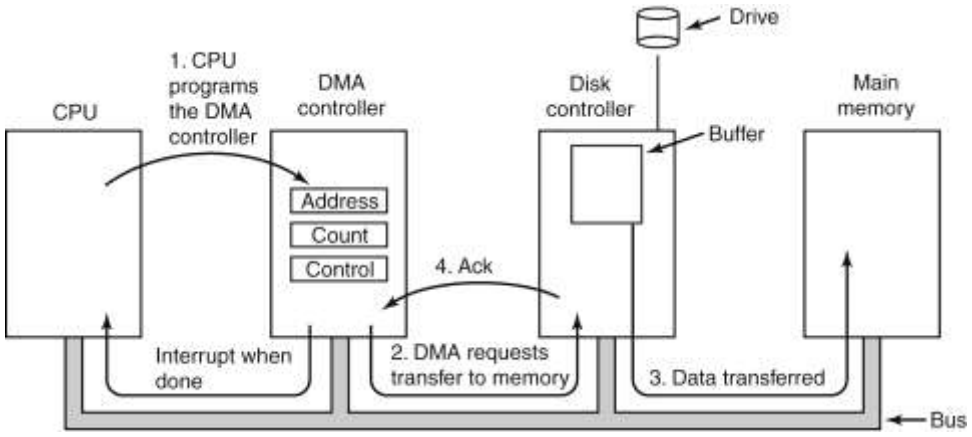
- Caching a device control register can be disastrous
- Hardware complications




Cache bus
Local bus
Memory bus
Level 2 cache
CPU
PCI bridge
Main memory
PCI bus
SCSI
USB
USB bus
ISA bridge
IDE disk
Graphics adaptor
Available PCI slot
SCSI bus
Mouse
Key-board
IDE bus
Mon-itor
ISA bus
Modem
Sound card
Printer
Available ISA slot

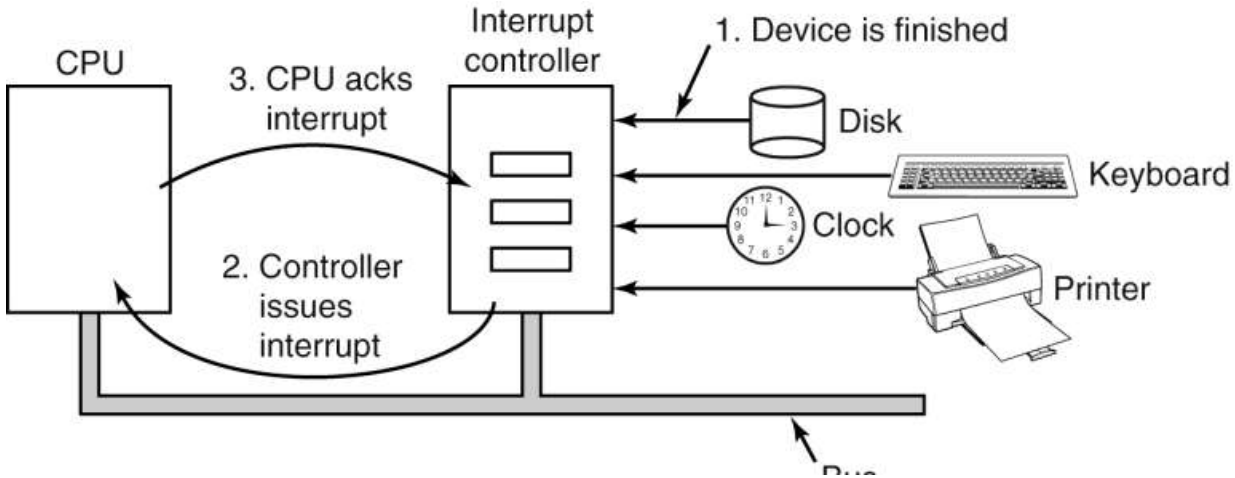
Direct Memory Access (DMA)

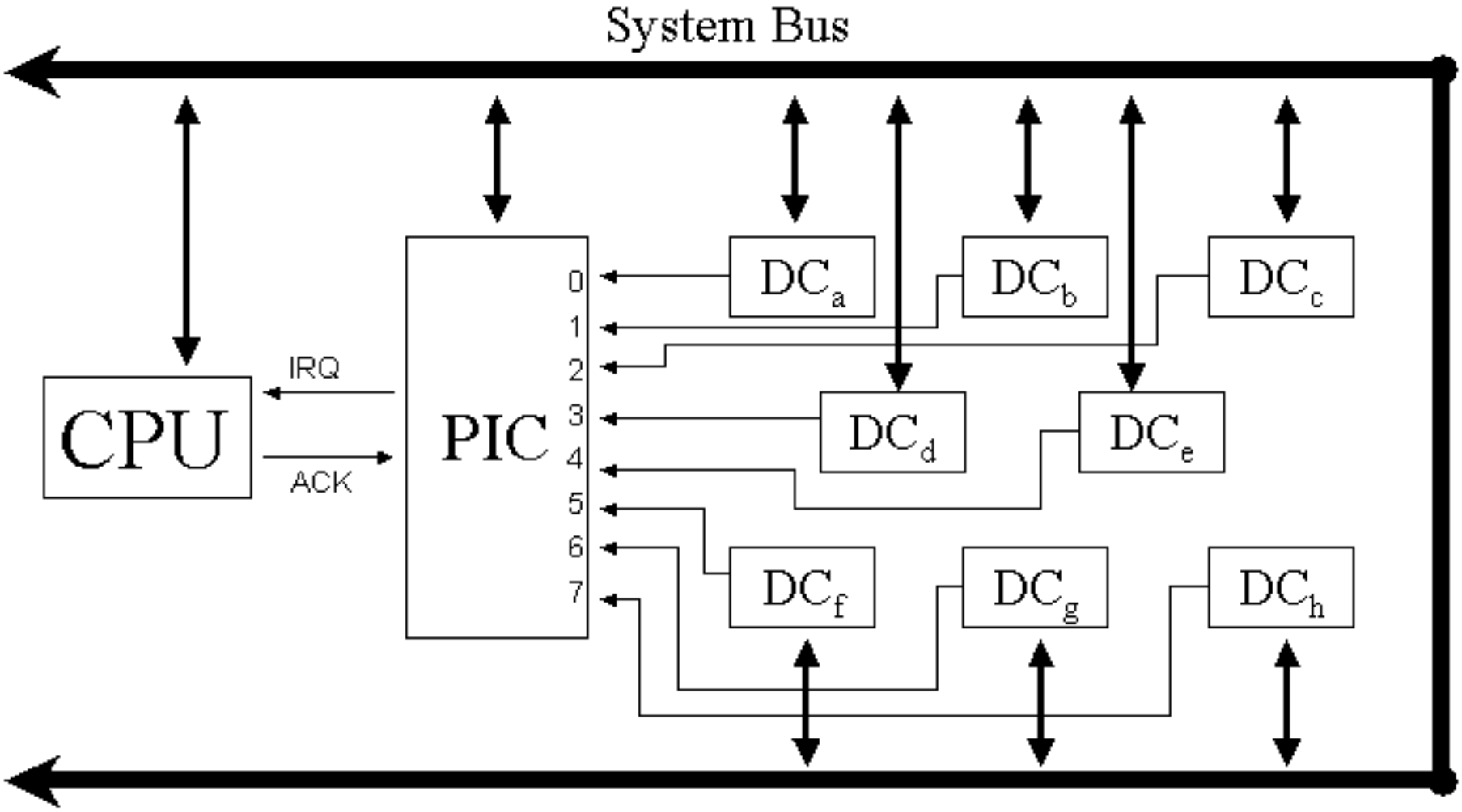
- It is not efficient for the CPU to request data from I/O one byte at a time
- DMA controller has access to the system bus independent of the CPU



Interrupts

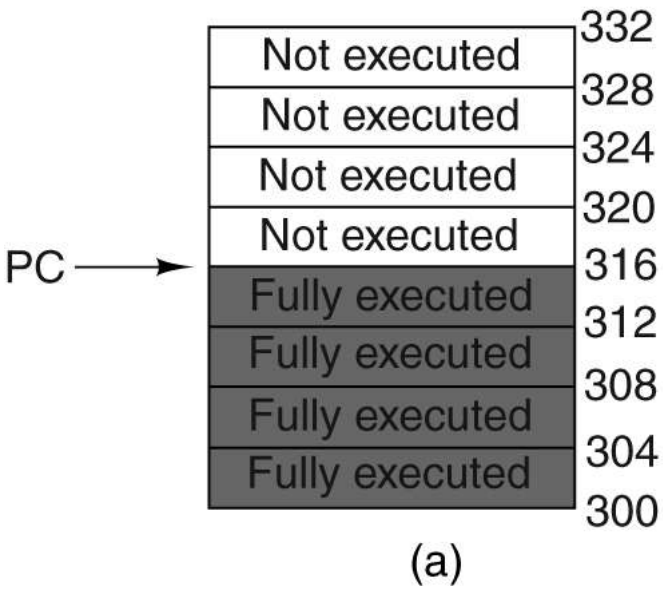
- When an I/O device has finished the work given to it, it causes an interrupt.



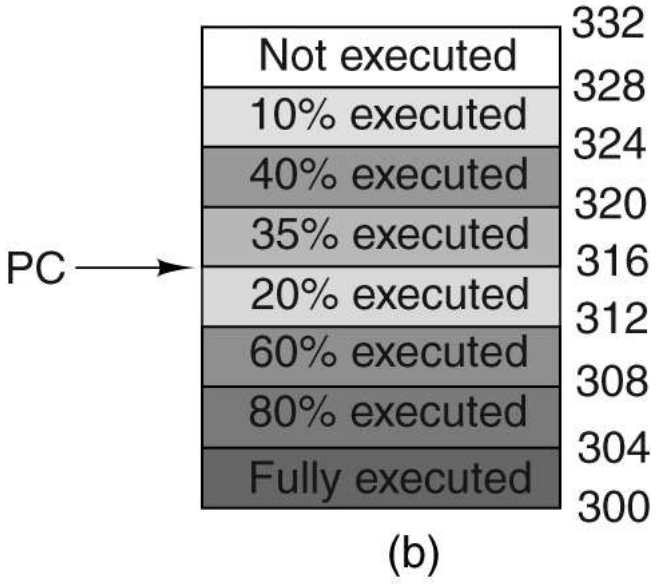

System Bus
CPU
IRQ
ACK
PIC
0
1
2
3
4
5
6
7
DCa
DCb
DCc
DCd
DCe
DCf
DCg
DCh

Precise Interrupts

- Makes handling interrupts much simplers
- Has 4 properties
  - The program counter (PC) is saved in known place
  - All instructions before the one pointed by PC have fully executed
  - No instruction beyond the one pointed by PC has been executed
  - The execution state of the instruction pointed to by the PC is known



Precise Interrupt



Imprecise Interrupt

I/O Software

- Device independence
- Uniform naming
- Error handling
  - Should be handled as close to the hardware as possible
- Synchronous vs asynchronous (interrupt-driven)
- Buffering
- Sharable verses dedicated devices

Three Ways of Doing I/O

- Programmed I/O
- Interrupt-driven I/O
- I/O Using DMA

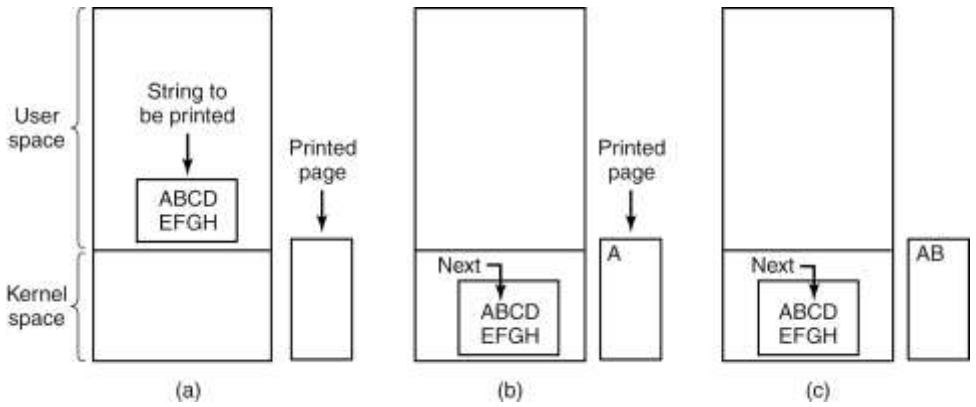
	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Programmed I/O

- CPU does all the work
- Busy-waiting (polling)

Example:

```
copy_from_user(buffer, p, count);                    /* p is the kernel buffer */
for (i = 0; i < count; i++) {                        /* loop on every character */
      while (*printer_status_reg != READY) ;     /* loop until ready */
      *printer_data_register = p[i];                 /* output one character */
}
return_to_user( );
```

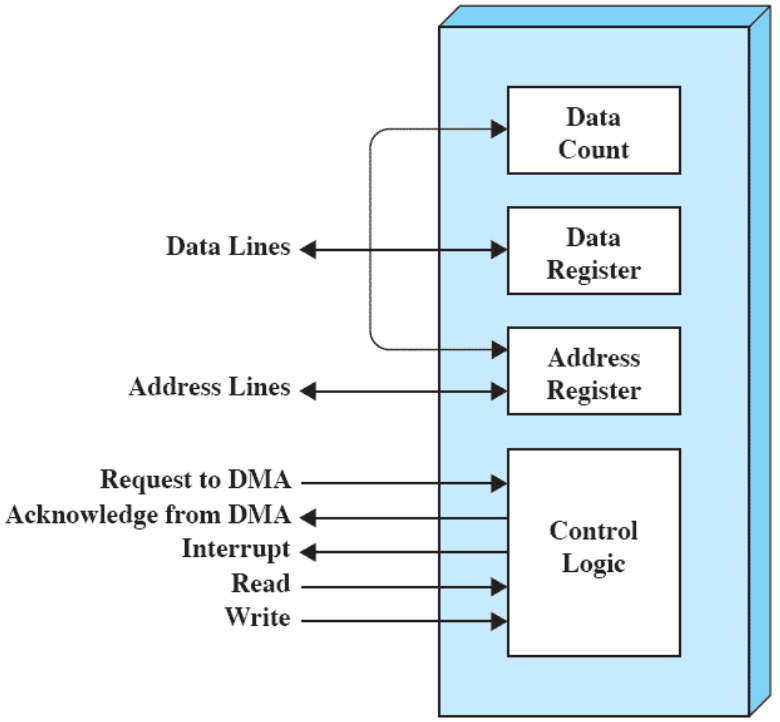


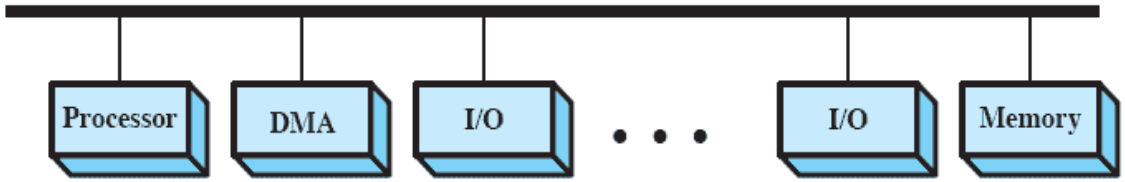
Interrupt-Driven I/O

- Waiting for a device to be ready, the process is blocked and another process is scheduled.
- When the device is ready it raises an interrupt.

I/O Using DMA

- DMA does the work instead of the CPU
- Let the DMA do its work and then interrupts

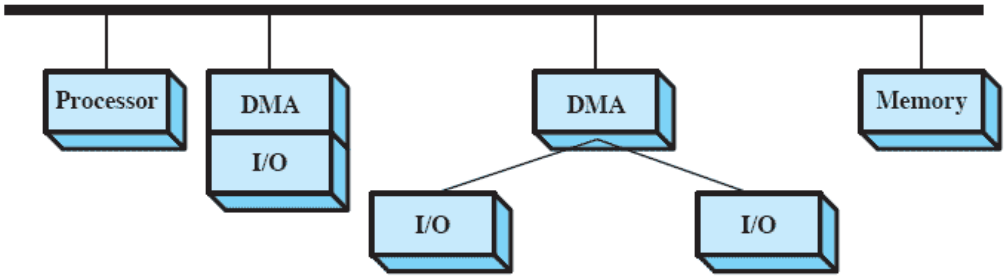




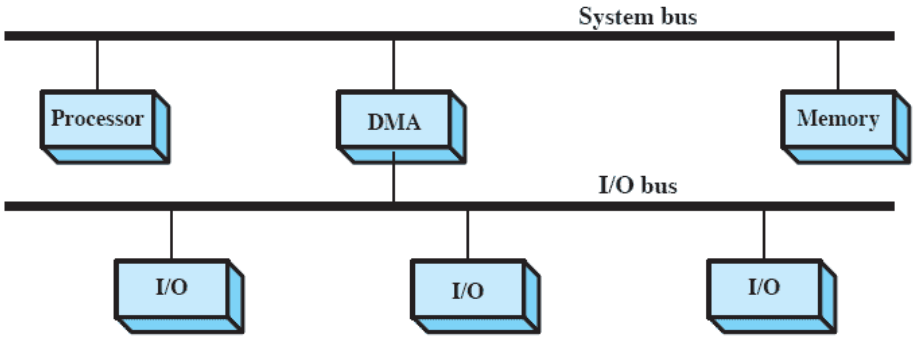
(a) Single-bus, detached DMA

DMA

Alternative



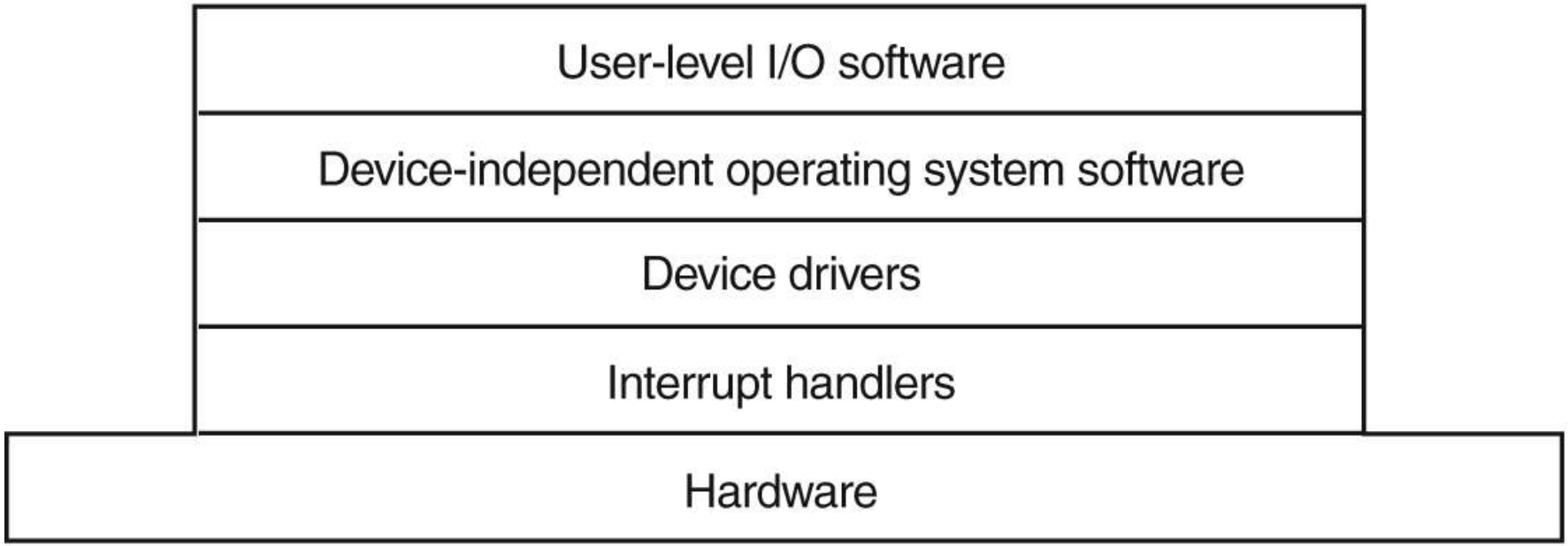
(b) Single-bus, Integrated DMA-I/O



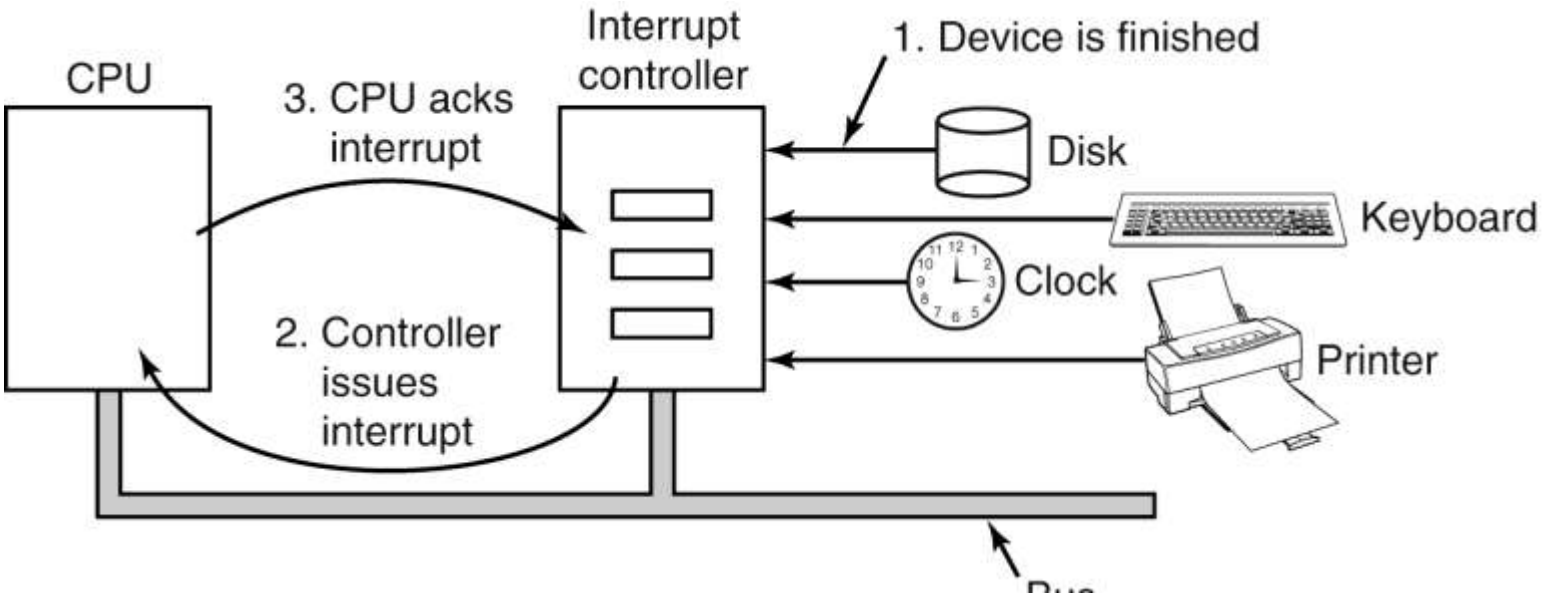
(c) I/O bus

Configurations

OS Software Layers for I/O



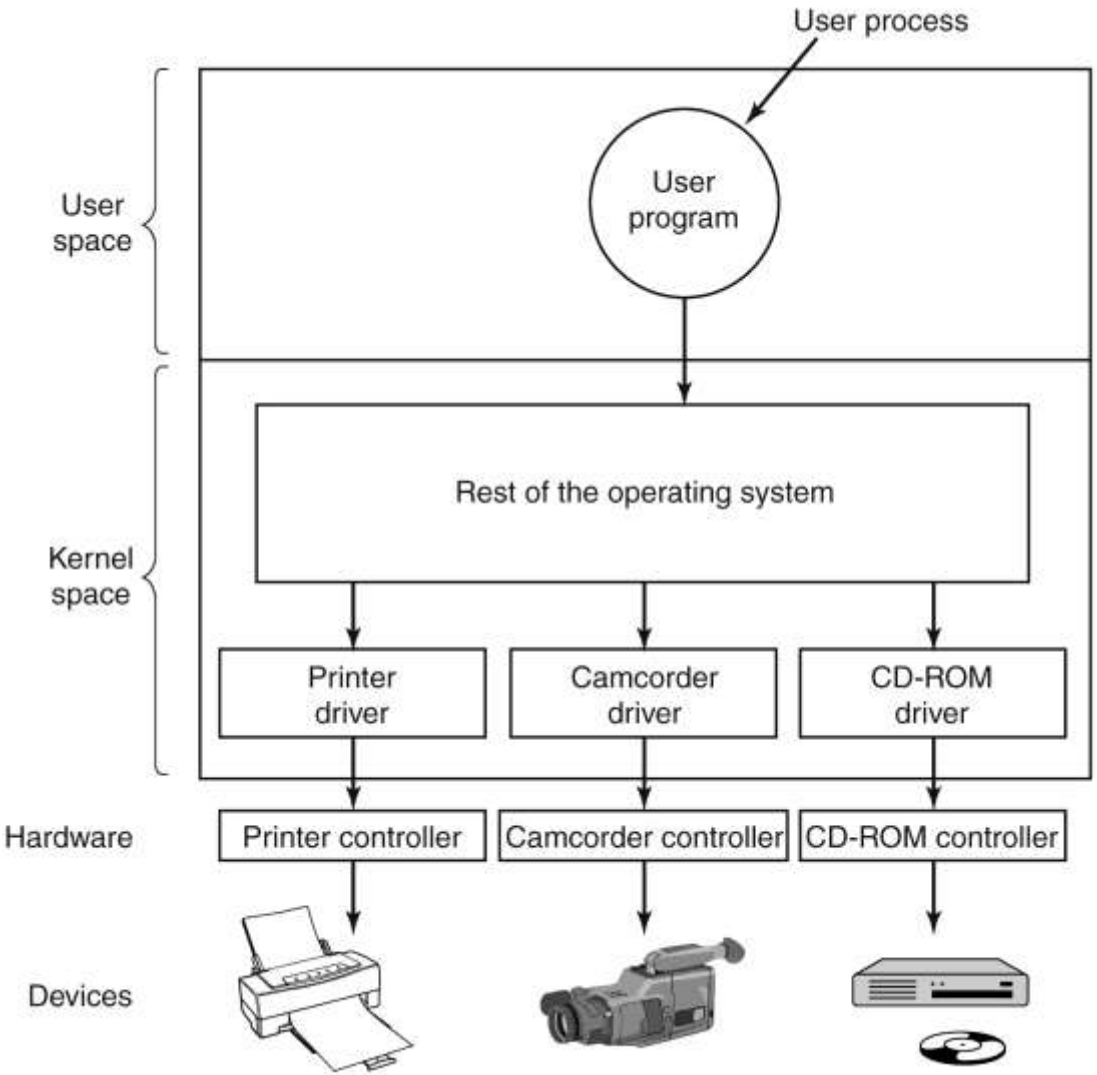
Interrupt Handlers



Device Drivers

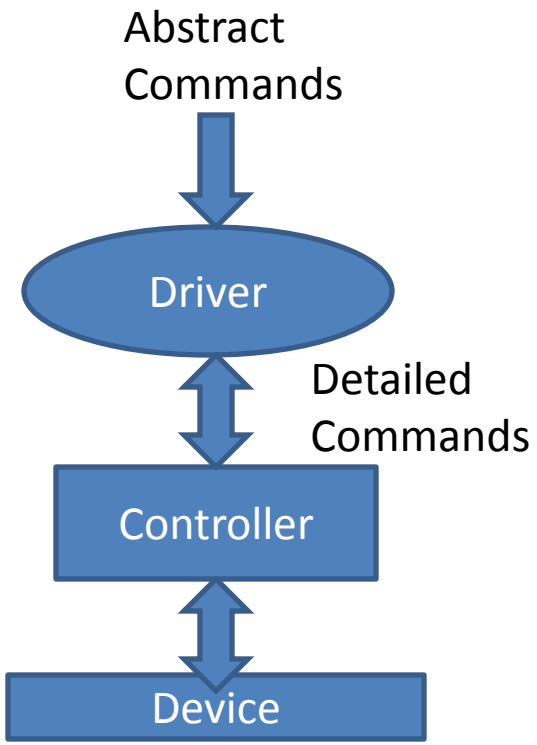
- Device specific code for controlling the device
  - Read device registers from controller
  - Write device registers to issue commands
- Usually supplied by the device manufacturer
- Can be part of the kernel or at user-space (with system calls to access controller registers)
- OS defines a standard interface that drivers for block devices must follow and another standard for driver of character devices

Device Drivers



Device Drivers

- Main functions:
  - Receive abstract read/write from layer above and carry them out
  - Initialize the device
  - Log events
  - Manage power requirements
- Drivers must be reentrant
- Drivers must deal with events such as a device removed or plugged



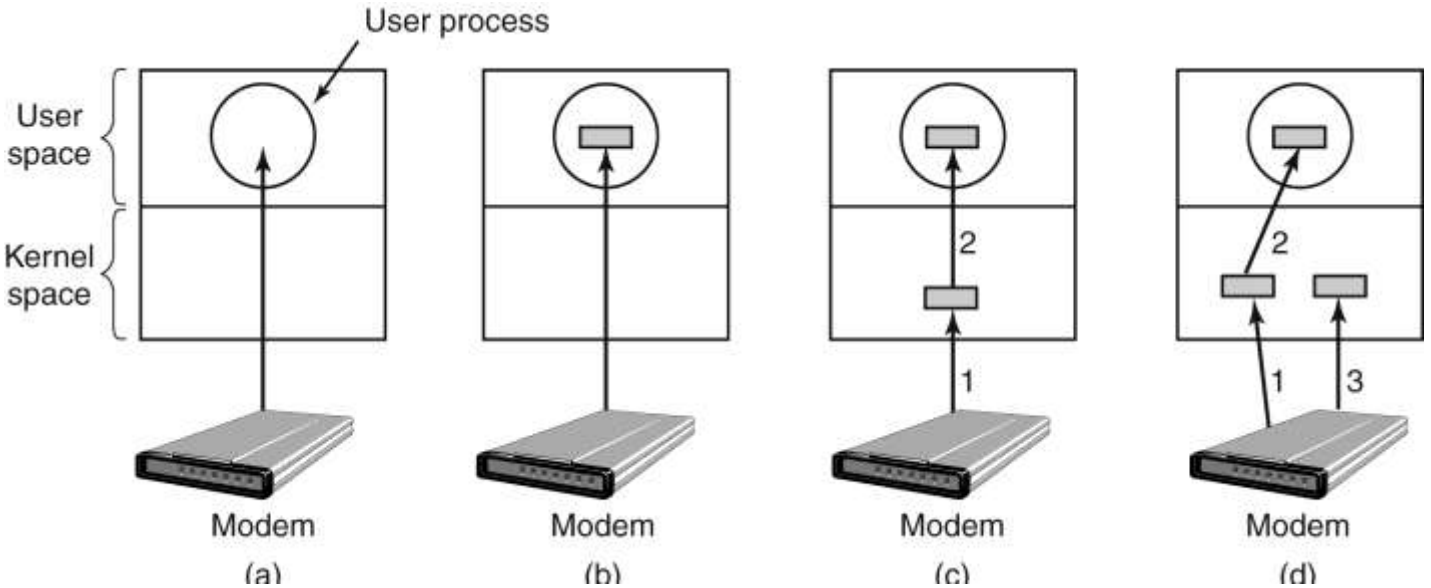
Device Independent I/O Software

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

Device Independent I/O Software

- Uniform interfacing for device drivers
  - Trying to make all devices look the same
  - For each class of devices, the OS defines a set of functions that the driver must supply.
  - This layer of OS maps symbolic device names onto proper drivers

Device Independent I/O Software



Interrupt with every character

Very inefficient

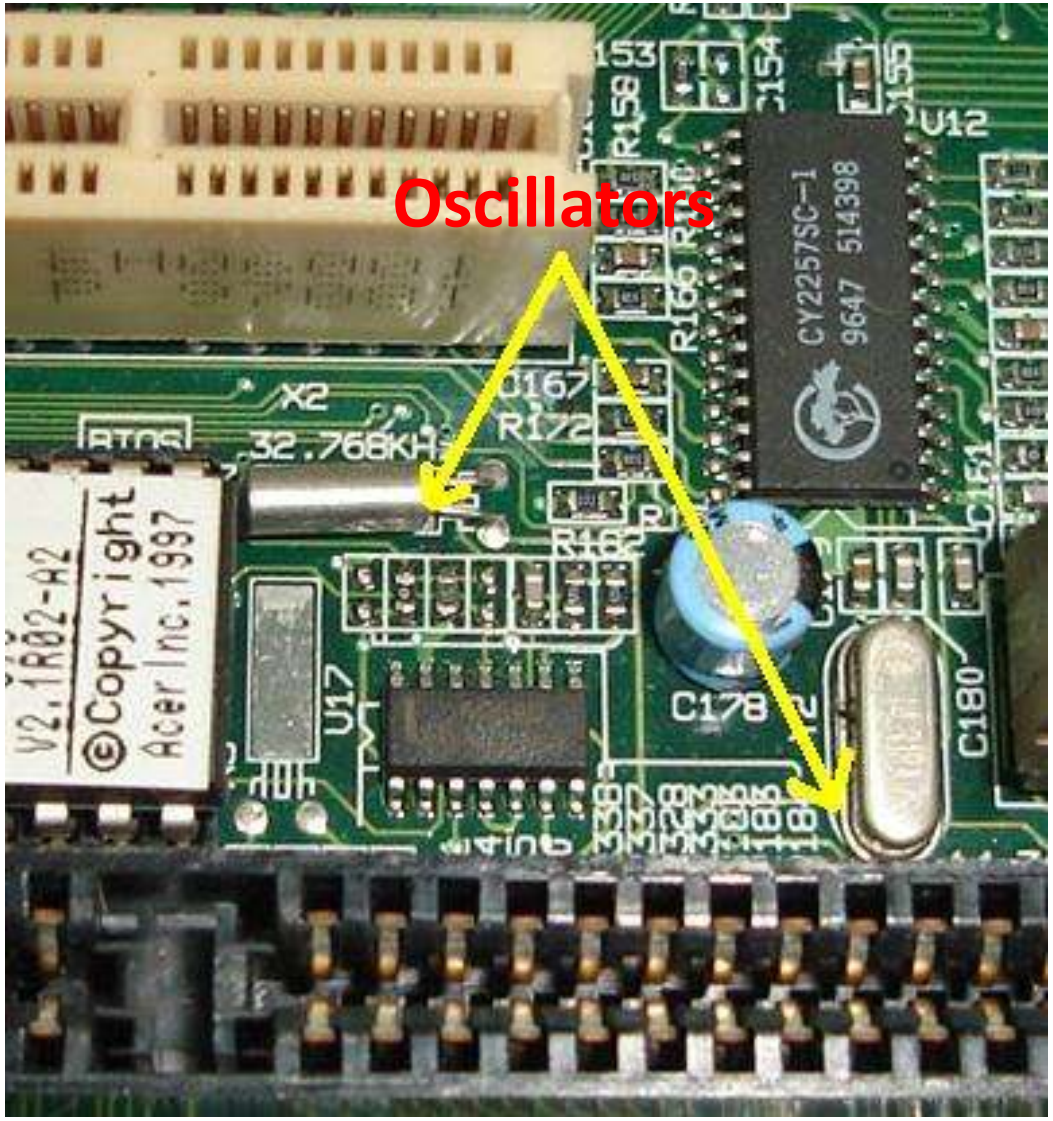
Buffering n char.

What is buffer is paged out?

The need for buffering

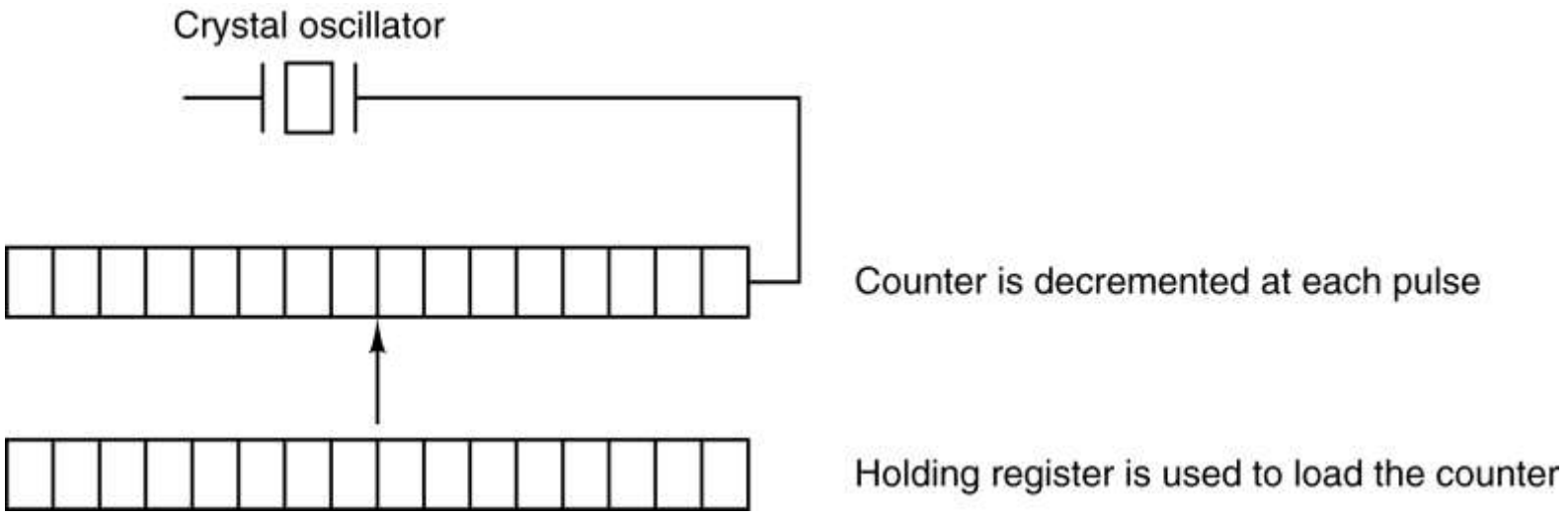
Example of Devices

Clocks


Oscillators
X2
32.768KH
R172
C167
U12
CY2257SC-1
9647 514398
C178
C180
U17
©Copyright
Acer Inc. 1997

Clock Hardware

- Old: tied to the power line and causes an interrupt on every voltage cycle
- New: Crystal oscillator + counter + holding register



Clock Software

- Maintaining the time of the day
- Preventing processes from running longer than they are allowed to
- Accounting for CPU usage
- Handling alarm system call made by user processes
- Providing watchdog timers for parts of the system itself
- Doing profiling, monitoring, and statistics gathering

User Interfaces

- Keyboard
- Mouse
- Monitor

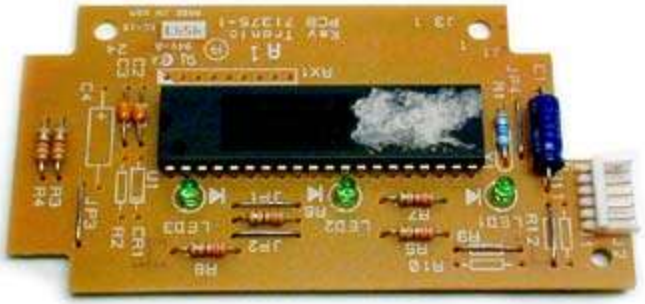
As examples, we will take a closer look at the keyboard and mouse.

Keyboards

- Contains simplified embedded processor that communicates through a port with the controller at the motherboard
- An interrupt is generated whenever a key is struck and a second one whenever a key is released
- Keyboard driver extracts the information about what happens from the I/O port assigned to the keyboard
- The number in the I/O port is called the scan code (7 bits for code + 1 bit for key press/release)

Keyboards

Microprocessor of the keyboard



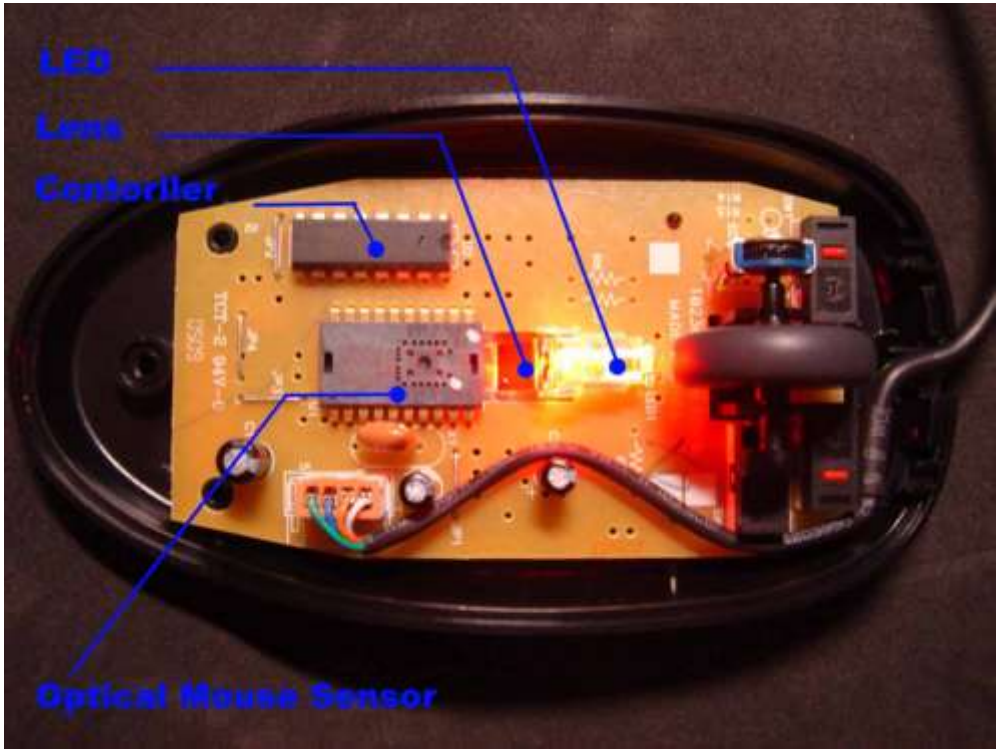
Key matrix

Keyboards

- There are two philosophies for programs dealing with keyboards
  1. The programs gets a raw sequence of ASCII codes (raw mode, or noncanonical mode)
  2. Driver handles all the intraline editing and just delivered corrected lines to the use programs (cooked mode or canonical mode)
- Either way, a buffer is needed to store characters

Mouse

- Mouse only indicates changes in position, not absolute position (delta_x and delta_y)



Conclusions

- The OS provides an interface between the devices and the rest of the system.
- The I/O part of the OS is divided into several layers.
- The hardware: CPU, programmable interrupt controller, DMA, device controller, and the device itself.
- OS must *expand* as new I/O devices are added