CSC 539: Operating Systems Structure and Design

Spring 2006

I/O management overview

- device controllers, interrupts and DMA
- disk scheduling
- reliability, RAID

protection & security overview

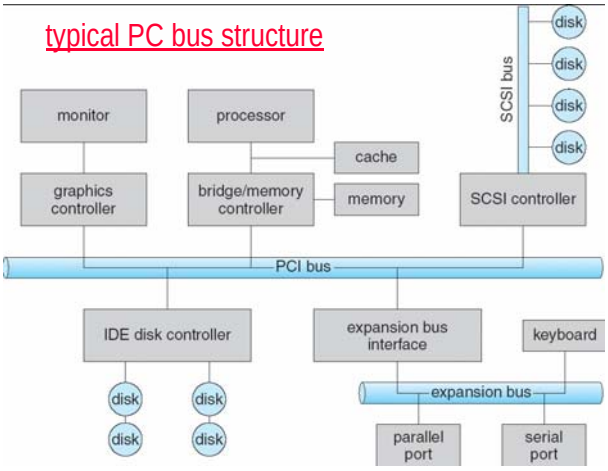
- protection: domain structure, access matrix
- security: authorization, program threats & system threats
- malware

Input/output systems

Ch 12 & 13 deal with I/O hardware, interrupts, characteristics of I/O

- much has been previously discussed, will focus on a few points

typical PC bus structure



Controllers

each controller has registers in it to receive both data and commands

- *status* : bits signal whether command is done, data is available, error?, …
- *control* : can be written by host (process requesting I/O) to set mode of input
- *data-in* : read by host to get input
- *data-out* : written by host to send output

the controller and host interact via *handshaking*

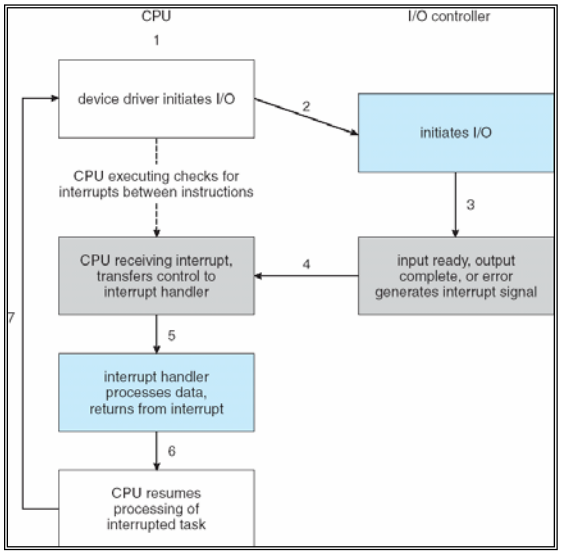
e.g., consider host process that wants to write data to disk.

for each byte:

1. host waits until disk is free (either polls *busy* bit of *status* register periodically or else relies on interrupt)
2. host sets *write* bit in *control* register, writes byte into *data-out* register
3. host sets *command-ready* bit in control register
4. when controller notices the *command-ready* bit is set, it sets the *busy* bit
5. controller reads control register & recognizes write command, reads *data-out* register and outputs data to the device

3

Interrupt-driven I/O cycle



CPU hardware has a wire called the *interrupt-request line*

CPU checks that wire after every instruction

if interrupt pending, jumps to the corresponding interrupt handling routine

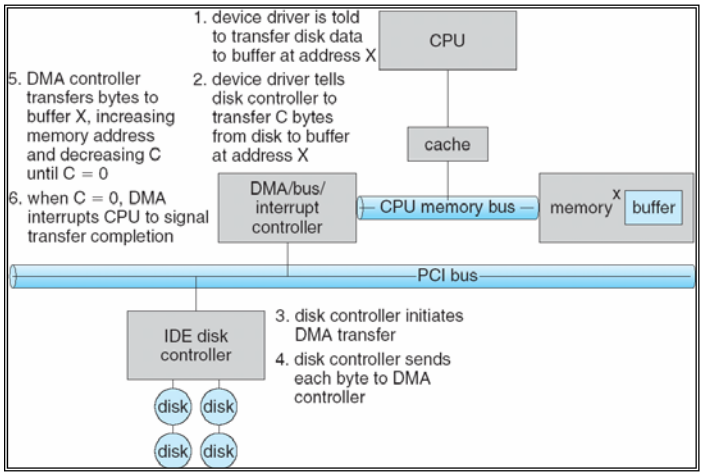
interrupt handler services the device (saving state of current process first)

4

Direct Memory Access (DMA)

for a device that transfers large amounts of data, e.g., disk drive, handshaking one byte at a time is wasteful

- can speed up transfer by allowing controller direct access to memory

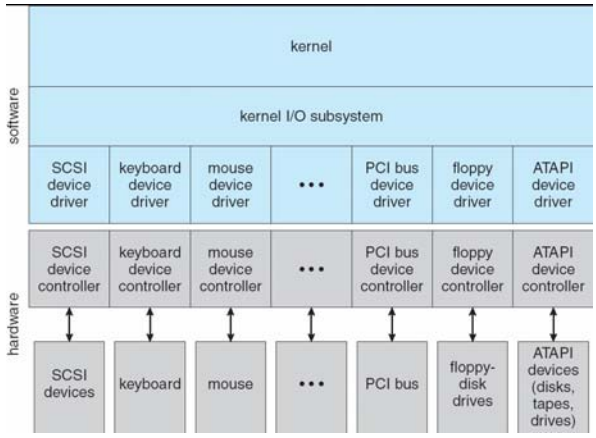


5

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
- devices vary in many dimensions
  character-stream or block, sequential or random-access, sharable or dedicated, ...



6

Disk scheduling

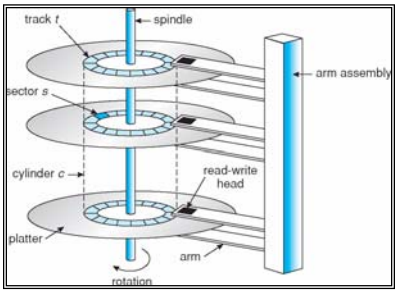
the OS is responsible for using hardware efficiently

- if the disk and controller are not busy when request arrives, handle immediately
- if not, then must save requests and schedule them

disk access time has two major components

- *seek time* is the time for the disk are to move the heads to the right track
- *rotational latency* is the additional time for the disk to rotate to the right sector

disk bandwidth =
amount transferred / time to completion



given a sequence of disk accesses, can schedule to maximize bandwidth (similar to how CPU scheduling maximized throughput)

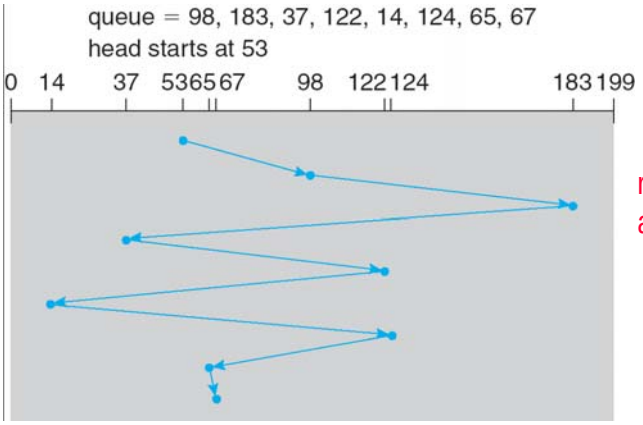
7

FCFS disk scheduling

first-come-first-served: schedule disk access requests in order they arrive

- simple, but not necessarily efficient use of read/write head

suppose read/write head is currently at track 53, and requests arrive:



requires movement across 640 cylinders

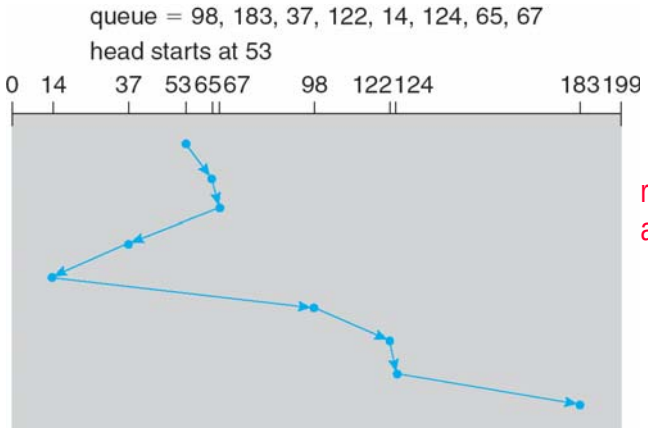
8

SSTF disk scheduling

shortest seek time first: schedule next request to minimize movement

- variation of SJF
- better than FCFS, but can lead to starvation

suppose read/write head is currently at track 53, and requests arrive:



requires movement across 236 cylinders

9

SCAN & LOOK disk scheduling

SCAN: start at one end and serve requests, then reverse

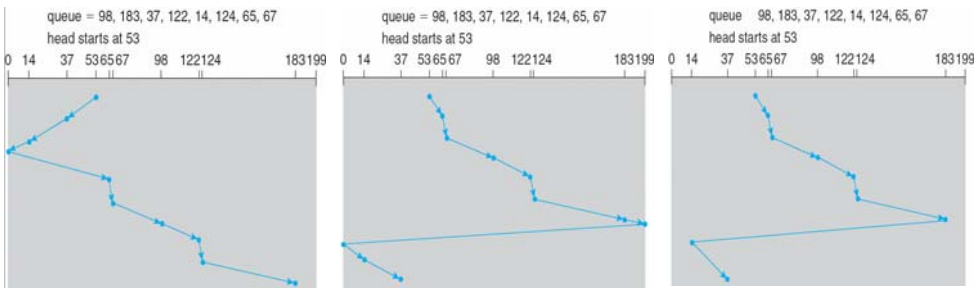
- a.k.a. the elevator algorithm, no starvation

C-SCAN: similar to SCAN, but rewinds when reaches end

- provides more consistent wait times

C-LOOK: similar to C-SCAN, but only go as far as most extreme requests

- attempts to avoid extraneous traversals at the extremes



10

Choosing a disk scheduling algorithm

- SSTF is common and has a natural appeal
- SCAN, C-SCAN, & C-LOOK perform better for systems that place a heavy load on the disk (no starvation)
- performance depends on the number and types of requests; can be influenced by the file-allocation method
- either SSTF or C-LOOK is a reasonable choice for the default algorithm

11

Swap-space management

in a system that uses swapping/paging, disk utilization is key to performance

- might use disk to store the entire process
- might use disk to only store pages that have been swapped out

TRADEOFFS?

- might store pages within the standard file system
- might create swap space in a separate partition, utilize separate storage manager

TRADEOFFS?

12

Disk reliability

head crashes were once quite common, 80's PC's crashed within 2-3 years

- tolerances have decreased, but technology and techniques have improved
- head crashes are much less likely today

- however, reliance on more disks increases chances of failure
  e.g. consider mean time to failure estimates

  mttf(1 disk) = 100,000 hours = 11.4 years

  mttf(100 disks) = 100,000/100 = 1,000 hours = 41.6 days

additional reliability can be obtained by greater use of redundant data and comprehensive error correcting codes (ECC)

13

RAID (Redundant Array of Independent Disks)

introduced in 1988 (originally as low-cost alternative to large disks)

mirroring : a logical disk consists of two physical disks

- every write is carried out on both disks, if one fails can read from other

  suppose mttf(1 disk) = 100,000 hours, takes 10 hours to fix/replace a disk
  mean time to data loss = $100{,}000^2/(2\times10)$ hours = 57,000 years

striping : a logical disk consists of multiple disks

- bits of each byte are spread across the disks (e.g., 8 disks, 1 bit per disk)
- since can access bits in parallel, provides faster access

- when used in conjunction with ECC can recover from single disk failure

14

RAID levels

level 0: block-level striping
- for high-performance systems where data loss is not critical

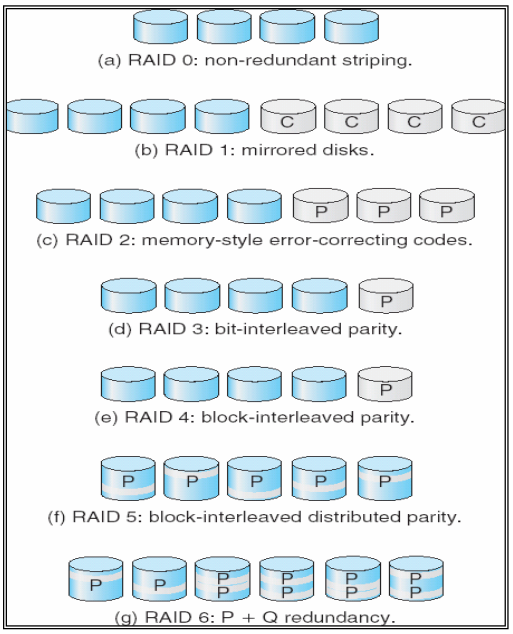
level 1: disk mirroring
- high reliability and fast data recovery
- but requires double the storage

level 2/3: bit-level striping + ECC
- with parity bit on extra disk, can recover from any single failure

level 4: block-level striping + ECC
- good for reading/writing large files (can process multiple blocks at once)
- small writes require writing block, checking parity, then writing parity block

level 5: block-level striping + mix ECC
- spreads load by storing different parity blocks on different disks

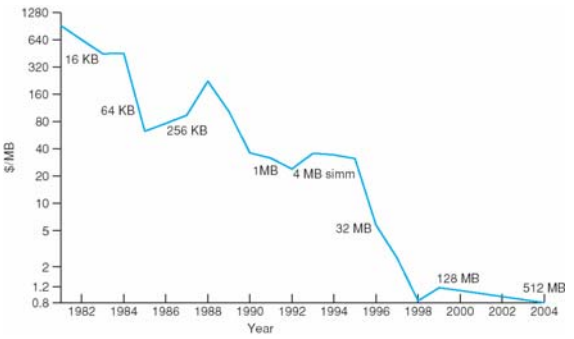
level 6: level 5 + redundancy
- allow recovery from multiple failures



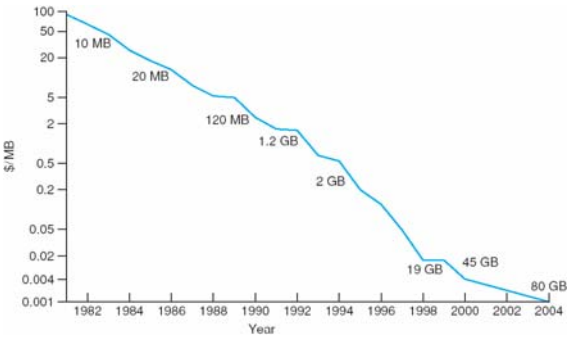
15

Cost comparisons

price per MB of DRAM, from 1981 to 2004



price per MB of Hard Disk, from 1981 to 2004



Protection

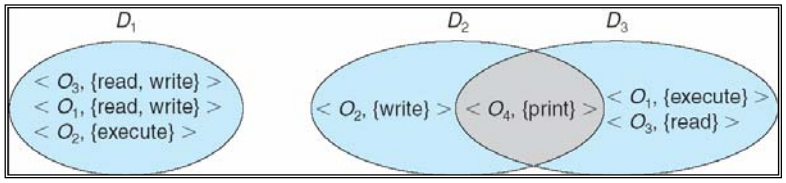
OS consists of a collection of objects, both hardware and software

- each object has a unique name, can be accessed through well-defined operations
- need to ensure that each object is accessed correctly & only by allowed processes

domain structure

access-right = <*object-name*, *rights-set*> where *rights-set* is subset of ops on that object

domain = set of access-rights



e.g., In UNIX, each user defines a new domain

Access matrix

can view protection as a matrix

- rows represent domains; columns represent objects
- `Access(i,j)` = set of ops that a process in $\texttt{Domain}_i$ can invoke on $\texttt{Object}_j$

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

could store each row as a *capability list* defining what operations are allowed for what objects within the domain

D_1 = <F_1, read>, <F_3, read>

could store each column as an *access list* defining who can perform what operations on the object

F_1 = <D_1, read>, <D_4, read+write>

Security

security is concerned with external environment, protection from:

- unauthorized access
- malicious modification or destruction
- accidental introduction of inconsistency

authorization is usually handled via passwords

- OS can help to ensure effectiveness/secrecy of passwords HOW?
  - ✓require non-dictionary passwords
  - ✓require frequent changes
  - ✓log all access attempts
  - ✓encrypt & hide passwords online
  - ✓biometrics

19

Program threats

Trojan horse

- code segment that misuses its environment.
  e.g. fake login script to steal passwords, shareware program with hidden agenda
  classic examples: NetBus, Back Orifice (BO), Back Orifice 2000 (BO2k)

trapdoor/backdoor

- specific user identifier or password that circumvents normal security procedures.
  e.g., War Games, almost in the 2003 Linux kernel
  classic examples: LiteBot, Remote Connection (RedNeck)

stack and buffer overflow

- exploits a bug in a program (overflow either the stack or memory buffers)
  classic examples: Internet worm (fingerd), Code Red (IIS), SQLSlammer (MS SQL Server)

spyware

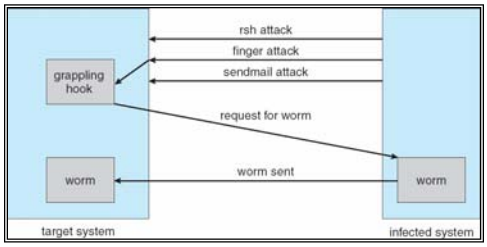
- rogue software that installs itself on a computer, reports personal info or activities
  e.g., adware (CoolWebSearch), stealware (180 Solutions)

20

System threats

worm

- standalone program that spawns copies, overwhelms the system
  e.g., Internet Worm (1988) – GAO estimated cost: $10M - $100M
  exploited UNIX networking features (*rsh*) and bugs in *finger* and *sendmail*



Robert Morris received 3 yrs probation, 400 hrs service, $10,000 fine

virus

- fragment of code embedded in a legitimate program
  e.g., Microsoft macro viruses

denial of service

- overload the targeted computer preventing it from doing any useful work

21

The cost of malware

Financial Impact of Virus Attacks 1995—2005

	Worldwide Impact (US $)
2005	$14.2 Billion
2004	17.5 Billion
2003	13.0 Billion
2002	11.1 Billion
2001	13.2 Billion
2000	17.1 Billion
1999	13.0 Billion
1998	6.1 Billion
1997	3.3 Billion
1996	1.8 Billion
1995	500 Million

Source: Computer Economics, 2006 *Figure 1*

DATE	EXPLOIT	TYPE	TARGET OF ATTACK
August 11	W32.Blaster.Worm	worm	Windows DCOM RPC
August 11	Backdoor.WinShell.50.b	trojan horse	Windows OS
August 12	W32.Randex.E	worm/trojan	Windows/Internet Relay Chat
August 12	W32.HLLW.Habrack	worm	Windows file sharing networks
August 13	W32.Blaster.B.Worm	worm	Windows DCOM RPC
August 13	W32.Blaster.C.Worm	worm	Windows DCOM RPC
August 13	VBS.Lembra@mm	worm	Microsoft Outlook
August 13	Backdoor.Beasty.H	trojan horse	Internet Explorer
August 14	Backdoor.Graybird.E	trojan horse	Windows security settings
August 14	W32.Kuskus.Worm	worm	Windows file sharing networks
August 14	W32.Randex.F	worm	Windows/Internet Relay Chat
August 14	W32.Randex.G	worm	Windows/Internet Relay Chat
August 15	W32.Bugsoft	worm	Microsoft Outlook
August 15	PWSteal.Lemir.C	trojan horse	Windows online games
August 15	Trojan.Analogx	trojan horse	Windows spoofed proxy server
August 16	W32.HLLW.SShydy.B	worm	Windows file sharing networks
August 16	W32.Randex.H	worm	Windows/Internet Relay Chat
August 16	W32.Dumaru@mm	worm/trojan	Windows/Internet Relay Chat
August 16	BAT.Randren	virus	Windows OS
August 18	W32.Welchia.Worm	worm	Windows DCOM RPC and IIS
August 18	W32.Dinkdink.Worm	worm	Windows DCOM RPC
August 18	**W32.Sobig.F@mm**	**worm**	**SMTP mass mailing worm**

week of 8/11/03 (CACM 12/03)

W32/Blaster-Lovsan

- worm that exploited buffer-overflow bug in Microsoft's RPC
- launched denial-of-service attack on Microsoft windowsupdate.com site
- contributed to Aug 14 blackout

SoBig

- worm that utilized email spoofing (tricks user into opening attachment)
- stored copy of itself on computer, steals addresses to try next
- accounted for 75% of Internet traffic at peak

22

Security solutions?

threat monitoring

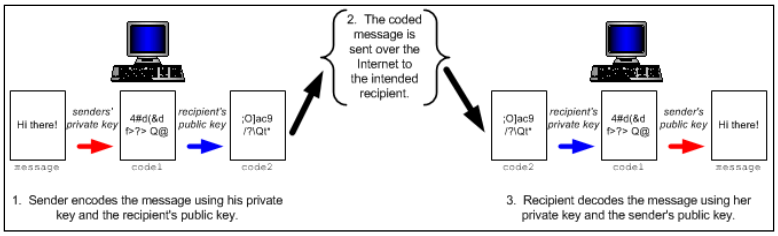
- check for suspicious patterns of activity (audit log)
- scan for security holes, apply patches religiously

firewall

- insert a machine between trusted and untrusted hosts to filter access
  e.g., Postini

use encryption where needed

- e.g., public key encryption and identify verification



23

Example: Windows XP

security is based on user accounts

- each user has unique security ID
- login to ID creates security access token
  - includes security ID for user, for user's groups, and special privileges every process gets copy of token
  - system checks token to determine if access allowed or denied

uses a *subject* model to ensure access security

- subject consists of an application + user's access token
- a subject tracks and manages permissions for each program that a user runs

each object in Windows XP has a security attribute defined by a *security descriptor*

- for example, a file has a security descriptor that indicates the access permissions for all users

24