CHAPTER 9: VIRTUAL MEMORY

- Background
- Demand Paging
- Performance of Demand Paging
- Page Replacement
- Page-Replacement Algorithms
- Allocation of Frames
- Thrashing
- Other Considerations
- Demand Segmentation

Background

- Virtual memory – separation of user logical memory from physical memory.
  - Only *part* of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Need to allow pages to be *swapped* in and out.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

Valid–Invalid bit

- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid bit is set to 0 on all entries.
- Example of a page table snapshot.

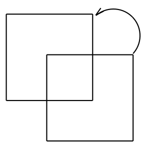
frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

Page Fault

1. If there is ever a reference to a page, first reference will trap to OS ⇒ *page fault*.
2. OS looks at another table to decide:
   a) Invalid reference ⇒ abort.
   b) Just not in memory.
3. Get empty frame.
4. Swap page into frame.
5. Reset tables, validation bit = 1.
6. Restart instruction:
   - block move



   - auto increment/decrement location

What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.

  ⇒ algorithm

  ⇒ performance – want an algorithm which will result in minimum number of page faults.

- Same page may be brought into memory several times.

Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  if $p = 0$, no page faults
  if $p = 1$, every reference is a fault
- Effective Access Time (EAT)
  EAT = $(1 - p) \times$ memory access
  + p (page fault overhead
  + [swap page out]
  + swap page in
  + restart overhead)
- Example:
  - memory access time = 1 microsecond
  - 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
  - Swap Page Time = 10 msec = 10,000 msec
  - EAT = $(1 - p) \times 1 + p$ (15000)
    = $1 + 15000P$ (in msec)

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Page-Replacement Algorithms

- Want lowest *page-fault rate.*
- Evaluate algorithm by running it on a particular string of memory references (*reference string*) and computing the number of page faults on that string.
- In all our examples, the reference string is

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

First-In-First-Out (FIFO) Algorithm

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3 frames (3 pages can be in memory at a time per process)

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

1	1	5	4	
2	2	1	5	
3	3	2		10 page faults
4	4	3		

FIFO Replacement – Belady's Anomaly

more frames $\not\Rightarrow$ less page faults

Optimal Algorithm

- Replace the page that will not be used for the longest period of time.
- 4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	4	
2		6 page faults
3		
4	5	

- How do you know this?
- Used for measuring how well your algorithm performs.

Least Recently Used (LRU) Algorithm

1		5
2		
3	5	4
4	3	

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be changed, look at the counters to determine which are to change
- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:

    move it to the top

    requires 6 pointers to be changed
  - No search for replacement

LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0.
  - When page is referenced bit set to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - Need reference bit.
  - Clock replacement.
  - If page to be replaced (in clock order) has reference bit = 1, then:

    a) set reference bit 0.

    b) leave page in memory.

    c) replace next page (in clock order), subject to same rules.

- Counting Algorithms – keep a counter of the number of references that have been made to each page.
  - LFU Algorithm: replaces page with smallest count.
  - MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
- Page-Buffering Algorithm – desired page is read into a free frame from the *pool* before the victim is written out.

Allocation of Frames

- Each process needs minimum number of pages.

  Example: IBM 370 – 6 pages to handle SS MOVE instruction:

  a) Instruction is 6 bytes, might span 2 pages.

  b) 2 pages to handle **from.**

  c) 2 pages to handle **to.**

- Two major allocation schemes:
  - fixed allocation
  - priority allocation

- Fixed allocation
  - Equal allocation

    If 100 frames and 5 processes, give each 20 pages.
  - Proportional allocation

    Allocate according to the size of process.
    - s_i = size of process p_i
    - $S = \Sigma s_i$
    - m = total number of frames
    - a_i = allocation for $p_i = \dfrac{s_i}{S} \times m$

Example : $m = 64$

$$s_1 = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

• Priority allocation

  - Use a proportional allocation scheme using priorities rather than size.

  - If process P_i generates a page fault,

    ◦ select for replacement one of its frames.

    ◦ select for replacement a frame from a process with lower priority number.

Global versus local allocation

• Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.

• Local replacement – each process selects from only its own set of allocated frames.

Thrashing

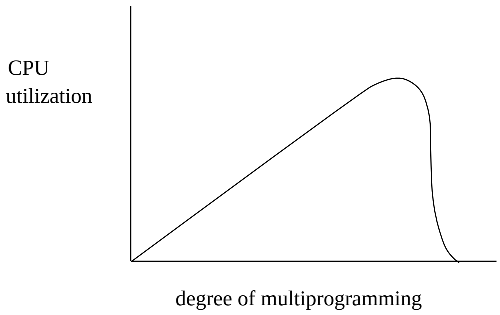
• If a process does not have ‘‘enough’’ pages, the page-fault rate is very high:

  ⇒ low CPU utilization.

  ⇒ operating system thinks that it needs to increase the degree of multiprogramming.

  ⇒ another process added to the system.

• Thrashing ≡ a process is busy swapping pages in and out.



• Why does paging work?

  Locality model

  - Process migrates from one locality to another.

  - Localities may overlap.

• Why does thrashing occur?

  Σ size of locality > total memory size

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references

  Example: 10,000 instruction

- WSS_i – working set of process P_i =

  total number of pages referenced in the most recent Δ (varies in time)

  If Δ too small will not encompass entire locality.

  If Δ too large will encompass several localities.

  If $\Delta = \infty \Rightarrow$ will encompass entire program.

- $D = \Sigma\, WSS_i \equiv$ total demand frames
- If $D > m \Rightarrow$ thrashing.
- Policy if $D > m$, then suspend one of the processes.

How do you keep track of the working set?

- Approximate with:

  interval timer + a reference bit

- Example:

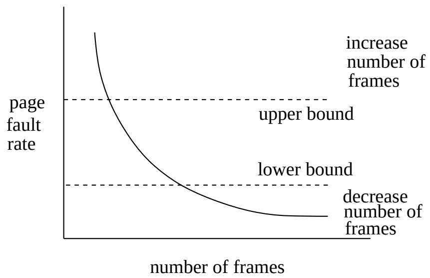
  $\Delta = 10{,}000$

  - Timer interrupts after every 5000 time units.
  - Keep in memory 2 bits for each page.
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set.

  Not completely accurate (why?)

  Improve = 10 bits and interrupt every 1000 time units

Page-Fault Frequency Scheme



- Establish ‘‘acceptable’’ page-fault rate.
  - If actual rate too low, process loses frame.
  - If actual rate too high, process gains frame.

Other Considerations

1. Prepaging
2. Page size selection
   - fragmentation
   - table size
   - I/O overhead
   - locality
3. Program structure
   - Array A[1024,1024] **of** integer
   - Each row is stored in one page
   - One frame
   - Program 1 **for** *j* := 1 to 1024 **do**
     for *i* := 1 to 1024 **do**
     A[*i* , *j*] := 0;

     1024×1024 page faults
   - Program 2 **for** *i* := 1 to 1024 **do**
     for *j* := 1 to 1024 **do**
     A[*i* , *j*] := 0;

     1024 page faults
4. I/O interlock and addressing

Demand Segmentation – used when insufficient hardware to implement demand paging.

- OS/2 allocates memory in segments, which it keeps track of through *segment descriptors.*
- Segment descriptor contains a valid bit to indicate whether the segment is currently in memory.
  - If segment is in main memory, access continues,
  - If not in memory, segment fault.

CHAPTER 10: FILE-SYSTEM INTERFACE

- File Concept
- Access Methods
- Directory Structure
- Protection
- Consistency Semantics

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
    - source
    - object (load image)
  - Documents

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file

Can simulate last two with first method by inserting appropriate control characters.

Who decides:
Operating system
Program

- File Attributes
  - **Name** – only information kept in human-readable form.
  - **Type** – needed for systems that support different types.
  - **Location** – pointer to file location on device.
  - **Size** – current file size.
  - **Protection** – controls who can do reading, writing, executing.
  - **Time, date, and user identification** – data for protection, security, and usage monitoring.
- Information about files are kept in the directory structure, which is maintained on the disk.

File Operations

- create
- write
- read
- reposition within file – file seek
- delete
- truncate
- open(F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory.
- close(F_i) – move the content of entry F_i in memory to directory structure on disk.

File Types – name.extension

File type	Usual extension	Function
Executable	exe, com, bin or none	ready-to-run machine-language program
Object	obj, o	compiled, machine language, not linked
Source code	c, p, pas, f77, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data, documents
Word processor	wp, tex, rrf, etc	various word-processor formats
Library	lib, a	libraries of routines for programmers
Print or view	ps, dvi, gif	ASCII or binary file in a format for printing or viewing
Archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage

Access Methods

- Sequential Access

  read next
  write next
  reset

  no *read* after last *write*

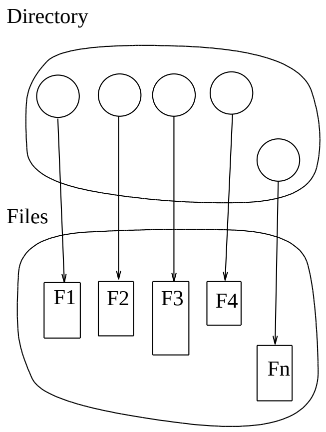
  (*rewrite*)

- Direct Access

  read n
  write n
  position to n
  read next
  write next
  rewrite n

n = relative block number

Directory Structure – a collection of nodes containing information about all files.



- Both the directory structure and the files reside on disk.
- Backups of these two structures are kept on tapes.

Information in a device directory:

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)

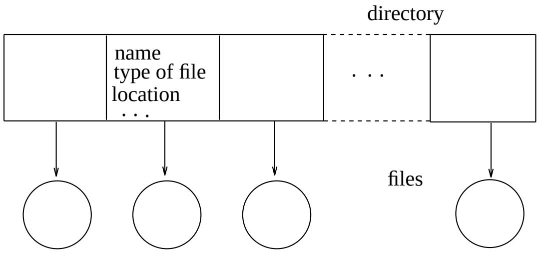
Operations performed on directory:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organize the directory (logically) to obtain:

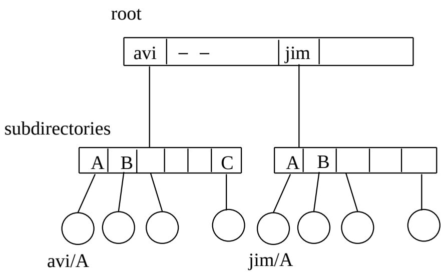
- Efficiency – locating a file quickly.
- Naming – convenient to users.
  - Two users can have same name for different files.
  - The same file can have several different names.
- Grouping – logical grouping of files by properties, e.g., all Pascal programs, all games, ...

Single-Level Directory – a single directory for all users.



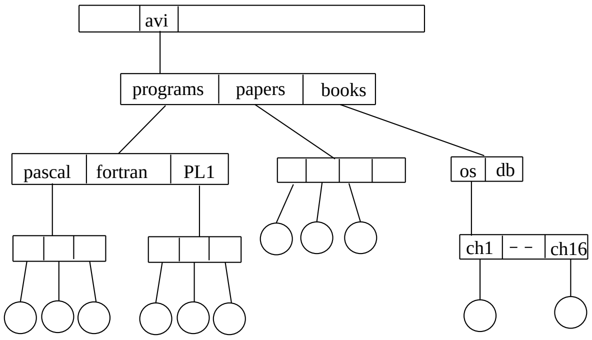
- Naming problem
- Grouping problem

Two-Level Directory – separate directory for each user.



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



- Efficient searching
- Grouping capability
- Current directory (working directory)

  cd /avi/books/os
  type ch1

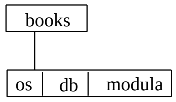
- Absolute or relative path name
- Creating a new file is done in current directory.
- Delete a file

  rm <file-name>
- Creating a new subdirectory is done in current directory.

  mkdir <dir-name>

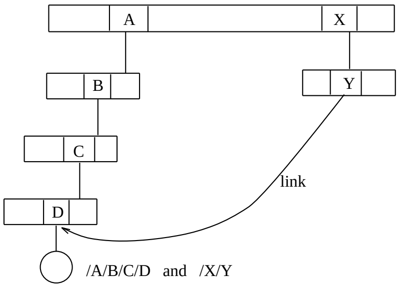
  Example: if in current directory /avi/books

  mkdir modula



- Deleting ‘‘books’’ ⇒ deleting the entire subtree rooted by ‘‘books’’.

Acyclic-Graph Directories – have shared subdirectories and files.



- Two different names (aliasing)
- If A deletes D ⇒ dangling pointer.

  Solutions:
  - Backpointers, so we can delete all pointers. Variable size records a problem.
  - Backpointers using a daisy chain organization.
  - Entry-hold-count solution.

General Graph Directory

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories.
  - Garbage collection.
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK.

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

				RWX
a)	owner access	7	⇒	1 1 1
b)	group access	6	⇒	1 1 0
c)	public access	1	⇒	0 0 1

- Ask manager to create a group (unique name), say *G*, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

owner group public

chmod 761 *game*

- Attach a group to a file

  chgrp *G* *game*

CHAPTER 11: FILE-SYSTEM IMPLEMENTATION

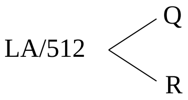
- File-System Structure
- Allocation Methods
- Free-Space Management
- Directory Implementation
- Efficiency and Performance
- Recovery

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks).
- File system organized into layers.
- *File control block* – storage structure consisting of information about a file.

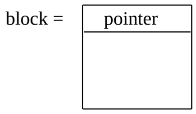
Contiguous Allocation – each file occupies a set of contiguous blocks on the disk.

- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.
- Mapping from logical to physical.



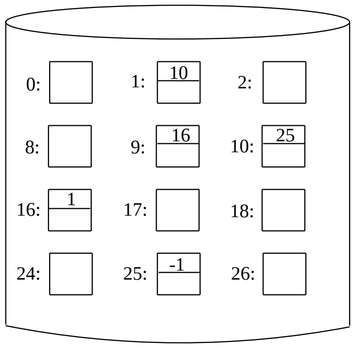
  - Block to be accessed = Q + starting address
  - Displacement into block = R

Linked Allocation – each file is a linked list of disk blocks; blocks may be scattered anywhere on the disk.



- Allocate as needed, link together.

  Example: File starts at block 9



Linked Allocation (continued)

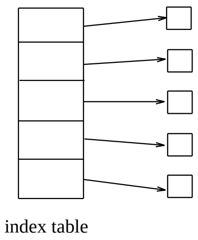
- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping

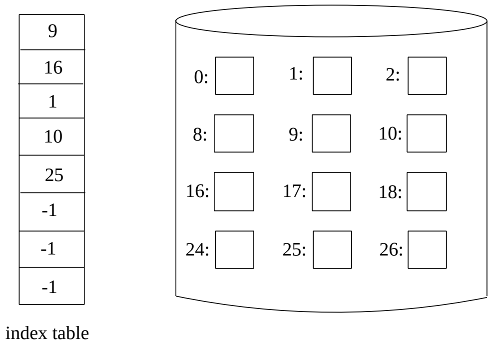
$$\text{LA}/511 < \begin{matrix} Q \\ R \end{matrix}$$

  - Block to be accessed is the Qth block in the linked chain of blocks representing the file.
  - Displacement into block = R + 1

- *File-allocation table (FAT)* – disk-space allocation used by MS-DOS and OS/2.

Indexed Allocation – brings all pointers together into the *index block*.





Indexed Allocation (continued)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.

$$\text{LA}/512 < \begin{matrix} Q \\ R \end{matrix}$$

  - Q = displacement into index table
  - R = displacement into block

Indexed Allocation – mapping (continued)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
  - Linked scheme – Link blocks of index tables (no limit on size).

$$\text{LA}/(512\times511) < \begin{matrix} Q_1 \\ R_1 \end{matrix}$$

    - Q_1 = block of index table
    - R_1 is used as follows:

$$R_1/512 < \begin{matrix} Q_2 \\ R_2 \end{matrix}$$

      - Q_2 = displacement into block of index table
      - R_2 = displacement into block of file

Indexed Allocation – mapping (continued)

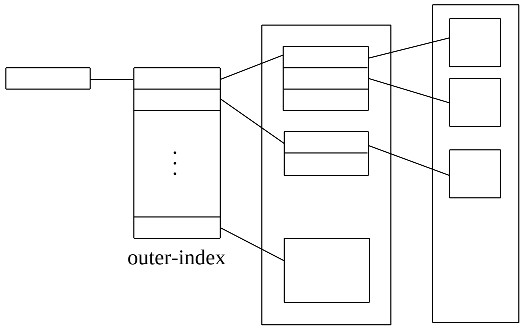
- Two-level index (maximum file size is 512^3)

$$\mathrm{LA}/(512\times512) < \begin{matrix} Q_1 \\ R_1 \end{matrix}$$

  - ◦ Q_1 = displacement into outer-index
  - ◦ R_1 is used as follows:

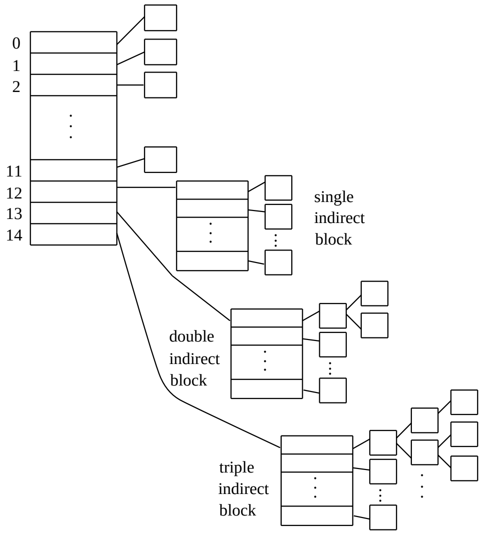
$$R_1/512 < \begin{matrix} Q_2 \\ R_2 \end{matrix}$$

    - ◦ Q_2 = displacement into block of index table
    - ◦ R_2 = displacement into block of file



Indexed Allocation – mapping (continued)

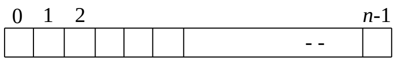
- Combined scheme: UNIX (4K bytes per block)



  - ◦ directly accessed 48K bytes
  - ◦ single indirection 2^{22} bytes
  - ◦ double indirection 2^{32} bytes

Free-Space Management

- Bit vector (*n* blocks)



$$\mathrm{bit}[i] = \begin{cases} 0 \Rightarrow \mathrm{block}[i] \text{ free} \\ 1 \Rightarrow \mathrm{block}[i] \text{ occupied} \end{cases}$$

  - Block number calculation

    (number of bits per word) ∗
    (number of 0-value words) +
    offset of first 1 bit

  - Bit map requires extra space.

    Ex: block size = 2^{12} bytes
    disk size = 2^{30} bytes (1 gigabyte)
    $n = 2^{30}/2^{12} = 2^{18}$

  - Easy to get contiguous files

- Need to protect:
  - Pointer to free list
  - Bit map
    - ◦ Must be kept on disk.
    - ◦ Copy in memory and disk may differ.
    - ◦ Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk.

    Solution:

    1) Set bit[*i*] = 1 in disk.
    2) Allocate block[*i*].
    3) Set bit[*i*] = 1 in memory.

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
- Counting

Directory Implementation

- Linear list of file names with pointers to the data blocks.
  - simple to program
  - time-consuming to execute
- Hash Table – linear list with hash data structure.
  - decreases directory search time
  - *collisions* – situations where two file names hash to the same location
  - fixed size

Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - *disk cache* – separate section of main memory for frequently used blocks
  - *free-behind* and *read-ahead* – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as *virtual disk*, or *RAM disk*

Recovery

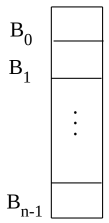
- Consistency checker – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by *restoring* data from backup.

CHAPTER 12: SECONDARY-STORAGE STRUCTURE

- Disk Structure
- Disk Scheduling
- Disk Management
- Swap-Space Management
- Disk Reliability
- Stable-Storage Implementation

Disk Structure

- A disk can be viewed as an array of blocks.



- There exists a mapping scheme from logical block address B_i to physical address (track, sector).
  - Smallest storage allocation area is a block.
  - Internal fragmentation on block.

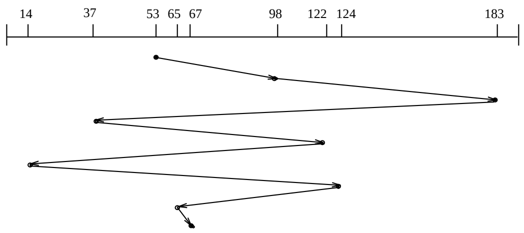
Disk Scheduling

- Disk Requests - Track/Sector
  - Seek
  - Latency
  - Transfer
- Minimize Seek Time
- Seek Time ≈ Seek Distance
- A number of different algorithms exist. We illustrate them with a request queue (0-199).

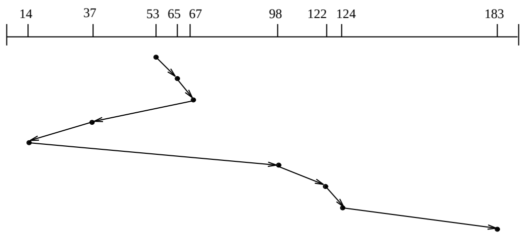
  98, 183, 37, 122, 14, 124, 65, 67

  Head pointer 53

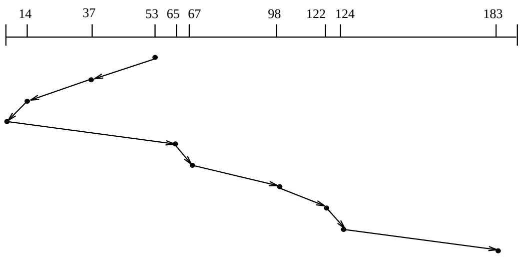
- FCFS



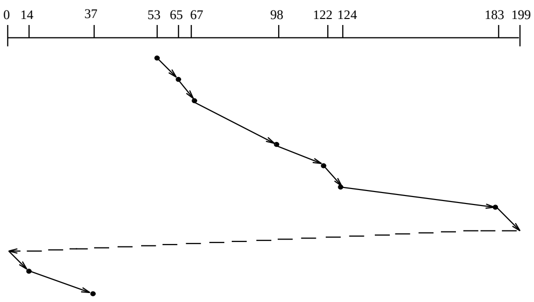
- SSTF



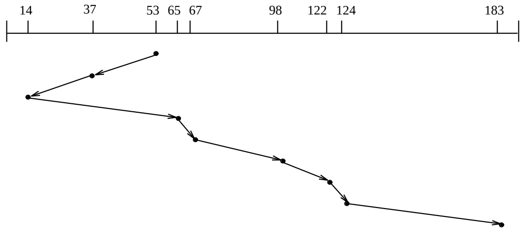
- SCAN



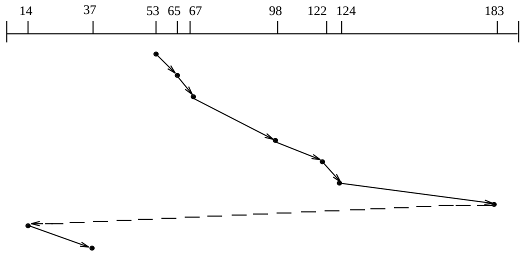
- C-SCAN



- LOOK



- C-LOOK



Disk Management

- Disk formatting
  - physical
  - logical
- Boot block initializes system.
- Need methods to detect and handle bad blocks.

Swap-Space Management

- Swap-space use
- Swap-space location
  - normal file system
  - separate disk partition
- Swap-space management
  - 4.3BSD allocates swap space when process starts (holds *text segment* and *data segment*).
  - Kernel uses *swap maps* to track swap-space use.

Disk Reliability

- Disk striping
- RAID
  - *Mirroring* or *shadowing* keeps duplicate of each disk.
  - *Block interleaved parity*.

Stable-Storage Implementation

- Write-ahead log scheme requires stable storage.
- To implement stable storage:
  - Replicate information on more than one non-volatile storage media with independent failure modes.
  - Update information in a controlled manner to ensure that failure during data transfer does not damage information.

CHAPTER 13: PROTECTION

- Goals of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection

Protection

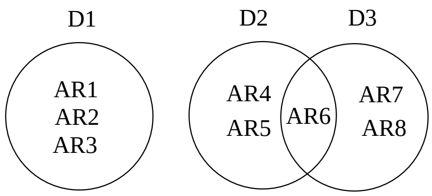
- Operating system consists of a collection of objects, hardware or software.
- Each object has a unique name and can be accessed through a well-defined set of operations.
- Protection problem – ensure that each object is accessed correctly and only by those processes that are allowed to do so.

Domain Structure

- Access-right = <object-name, rights-set>

  Rights-set is a subset of all valid operations that can be performed on the object.

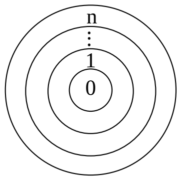
- Domain = set of access-rights



AR1 = <file_A, {Read,Write}>
AR8 = <file_A, {Read}>

Domain Implementation

- System consists of 2 domains:
  - user
  - supervisor
- UNIX
  - Domain = user-id
  - Domain switch accomplished via file system.
    - Each file has associated with it a domain bit (*setuid bit*).
    - When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset.
- Multics Rings
  - Let D_i and D_j be any two domain rings. If $j < i \Rightarrow D_i \subseteq D_j$.
  -



Access Matrix

- Rows – domains
- Columns – domains + objects
- Each entry – Access rights
  Operator names

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

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix.
- Can be expanded to dynamic protection.
  - Operations to add, delete access rights.
  - Special access rights:
    - *owner* of O_i
    - *copy* op from O_i to O_j
    - *control* – switch from domain D_i to D_j

- Access matrix design separates mechanism from policy.
  - Mechanism – operating system provides Access-matrix + rules.

    It ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.
  - Policy – user dictates policy.

    Who can access what object and in what mode.

Implementation of Access Matrix

- Each column = Access-control list for one object

  Defines who can perform what operation.

  Domain 1 = Read,Write
  Domain 2 = Read
  Domain 3 = Read
  .
  .
  .

- Each Row = Capability List (like a key)

  For each domain, what operations allowed on what objects.

  Object 1 – Read
  Object 4 – Read,Write,Execute
  Object 5 – Read,Write,Delete,Copy

Revocation of Access Rights

- Access List – Delete access rights from access list.
  - simple
  - immediate
- Capability List – Scheme required to locate capability in the system before capability can be revoked.
  - Reacquisition
  - Back-pointers
  - Indirection
  - Keys

Capability-Based Systems

- Hydra
  - Fixed set of access rights known to and interpreted by the system.
  - Interpretation of user-defined rights performed solely by user’s program; system provides access protection for the use of these rights.
- Cambridge CAP System
  - *Data capability* – provides standard read, write, execute of individual storage segments associated with object.
  - *Software capability* –interpretation left to the subsystem, through its protected procedures.

Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.

CHAPTER 14: SECURITY

- The Security Problem
- Authentication
- Program Threats
- System Threats
- Threat Monitoring
- Encryption

The Security Problem

- Security must consider external environment of the system, and protect it from:
  - unauthorized access.
  - malicious modification or destruction.
  - accidental introduction of inconsistency.
- Easier to protect against accidental than malicious misuse.

Authentication

- User identity most often established through *passwords*, can be considered a special case of either keys or capabilities.
- Passwords must be kept secret.
  - Frequent change of passwords.
  - Use of ‘‘non-guessable’’ passwords.
  - Log all invalid access attempts.

Program Threats

- Trojan Horse
  - Code segment that misuses its environment.
  - Exploits mechanisms for allowing programs written by users to be executed by other users.
- Trap Door
  - Specific user identifier or password that circumvents normal security procedures.
  - Could be included in a compiler.

System Threats

- Worms – use spawn mechanism; standalone program.
- Internet worm
  - Exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs.
  - Grappling hook program uploaded main worm program.
- Viruses – fragment of code embedded in a legitimate program.
  - Mainly effect microcomputer systems.
  - Downloading viral programs from public bulletin boards or exchanging floppy disks containing an infection.
  - *Safe computing.*

Threat Monitoring

- Check for suspicious patterns of activity – i.e., several incorrect password attempts may signal password guessing.
- Audit log – records the time, user, and type of all accesses to an object; useful for recovery from a violation and developing better security measures.
- Scan the system periodically for security holes; done when the computer is relatively unused. Check for:
  - Short or easy-to-guess passwords
  - Unauthorized set-uid programs
  - Unauthorized programs in system directories
  - Unexpected long-running processes
  - Improper directory protections
  - Improper protections on system data files
  - Dangerous entries in the program search path (Trojan horse)
  - Changes to system programs; monitor checksum values

Encryption

- Encrypt *clear text* into *cipher text.*
- Properties of good encryption technique:
  - Relatively simple for authorized users to encrypt and decrypt data.
  - Encryption scheme depends not on the secrecy of the algorithm but on a parameter of the algorithm called the *encryption key*.
  - Extremely difficult for an intruder to determine the encryption key.
- *Data Encryption Standard* substitutes characters and rearranges their order on the basis of an encryption key provided to authorized users via a secure mechanism. Scheme only as secure as the mechanism.

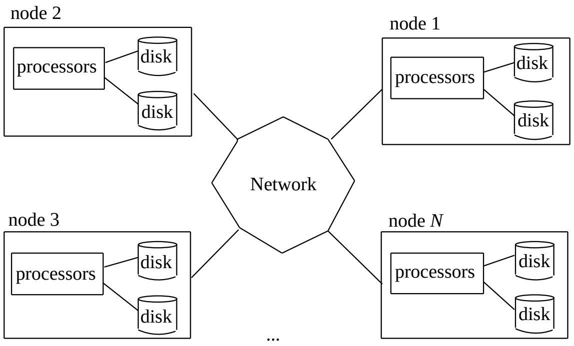
- *Public-key encryption* based on each user having two keys:
  - *public key* – published key used to encrypt data.
  - *private key* – key known only to individual user used to decrypt data.
- Must be an encryption scheme that can be made public without making it easy to figure out the decryption scheme.
  - Efficient algorithm for testing whether or not a number is prime.
  - No efficient algorithm is known for finding the prime factors of a number.

CHAPTER 15: NETWORK STRUCTURES

- Background
- Motivation
- Topology
- Network Types
- Communication
- Design Strategies

Background – general structure

- Nodes
  - Central processing unit
  - Main memory
  - Locally-attached disks
- Network



Background – node types

- Mainframes (IBM3090, etc.)
  - example applications:
    - airline reservations
    - banking systems
  - many large attached disks
- Workstations (Sun, Apollo, Microvax, RISC6000, etc.)
  - example applications:
    - computer-aided design
    - office-information systems
    - private databases
  - zero, one or two medium size disks
- Personal Computers
  - example applications:
    - office information systems
    - small private databases
  - zero or one small disk

Motivation

- Resource sharing
  - sharing and printing files at remote sites
  - processing information in a distributed database
  - using remote specialized hardware devices
- Computation speedup – *load sharing*
- Reliability – detect and recover from site failure, function transfer, reintegrate failed site
- Communication – message passing

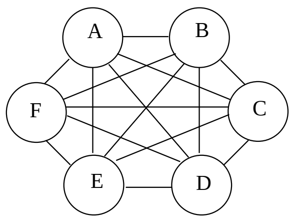
Topology

The sites in the system can be physically connected in a variety of ways. They are compared with respect to the following criteria:

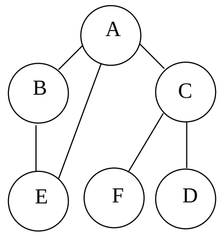
- **Basic cost**. How expensive is it to link the various sites in the system?
- **Communication cost**. How long does it take to send a message from site *A* to site *B*?
- **Reliability**. If a link or a site in the system fails, can the remaining sites still communicate with each other?

The various topologies are depicted as graphs whose nodes correspond to sites. An edge from node *A* to node *B* corresponds to a direct connection between the two sites.

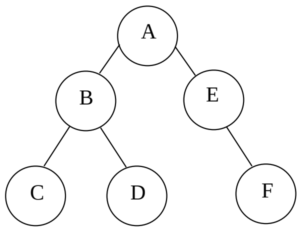
Topology (continued)



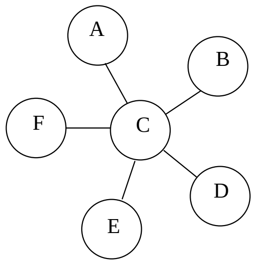
fully connected network



partially connected network

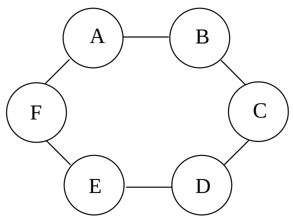


tree structured network

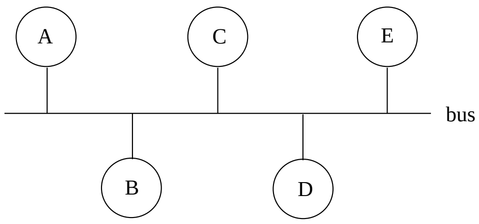


star network

Topology (continued)



ring network



shared bus network (Ethernet)

Network Types

- Local-Area Network (LAN) – designed to cover small geographical area.
  - Multiaccess bus, ring, or star network.
  - Speed ˜ 10 megabits/second, or higher.
  - Broadcast is fast and cheap.
  - Nodes:
    - usually workstations and/or personal computers
    - a few (usually one or two) mainframes
- Wide-Area Network (WAN) – links geographically separated sites.
  - Point-to-point connections over long-haul lines (often leased from a phone company).
  - Speed ˜ 100 kilobits/second.
  - Broadcast usually requires multiple messages.
  - Nodes:
    - usually a high percentage of mainframes

Communication

The design of a *communication* network must address four basic issues:

- **Naming and name resolution:** How do two processes locate each other to communicate?
- **Routing strategies**. How are messages sent through the network?
- **Connection strategies**. How do two processes send a sequence of messages?
- **Contention**. The network is a shared resource, so how do we resolve conflicting demands for its use?

Naming and Name Resolution

- Name systems in the network.
- Address messages with the process-id.
- Identify processes on remote systems by

  <host-name, identifier> pair.
- *Domain name service* (*DNS*) – specifies the naming structure of the hosts, as well as name to address resolution (Internet).

Routing Strategies

- **Fixed routing**. A path from *A* to *B* is specified in advance and does not change unless a hardware failure disables this path.
  - Since the shortest path is usually chosen, communication costs are minimized.
  - Fixed routing cannot adapt to load changes.
  - Ensures that messages will be delivered in the order in which they were sent.
- **Virtual circuit**. A path from *A* to *B* is fixed for the duration of one *session*. Different sessions involving messages from *A* to *B* may have different paths.
  - Partial remedy to adapting to load changes.
  - Ensures that messages will be delivered in the order in which they were sent.

Routing Strategies (continued)

- **Dynamic routing**. The path used to send a message from site *A* to site *B* is chosen only when a message is sent.
  - Usually a site sends a message to another site on the link least used at that particular time.
  - Adapts to load changes by avoiding routing messages on heavily used path.
  - Messages may arrive out of order. This problem can be remedied by appending a sequence number to each message.

Connection Strategies

- **Circuit switching**. A permanent physical link is established for the duration of the communication (i.e., telephone system).

- **Message switching**. A temporary link is established for the duration of one message transfer (i.e., post-office mailing system).

- **Packet switching**. Messages of variable length are divided into fixed-length packets which are sent to the destination. Each packet may take a different path through the network. The packets must be reassembled into messages as they arrive.

Circuit switching requires setup time, but incurs less overhead for shipping each message, and may waste network bandwidth. Message and packet switching require less setup time, but incur more overhead per message.

Contention

Several sites may want to transmit information over a link simultaneously. Techniques to avoid repeated collisions include:

- CSMA/CD. *Carrier sense with multiple access* (CSMA); *collision detection* (CD)
  - A site determines whether another message is currently being transmitted over that link. If two or more sites begin transmitting at exactly the same time, then they will register a CD and will stop transmitting.
  - When the system is very busy, many collisions may occur, and thus performance may be degraded.

  CSMA/CD is used successfully in the Ethernet system, the most common network system.

Contention (continued)

- **Token passing**. A unique message type, known as a *token*, continuously circulates in the system (usually a ring structure). A site that wants to transmit information must wait until the token arrives. When the site completes its round of message passing, it retransmits the token. A token-passing scheme is used by the IBM and Apollo systems.

- **Message slots**. A number of fixed-length message slots continuously circulate in the system (usually a ring structure). Since a slot can contain only fixed-sized messages, a single logical message may have to be broken down into a number of smaller packets, each of which is sent in a separate slot. This scheme has been adopted in the experimental Cambridge Digital Communication Ring.

Design Strategies

The communication network is partitioned into the following multiple layers:

- Physical layer – handles the mechanical and electrical details of the physical transmission of a bit stream.

- Data-link layer – handles the *frames*, or fixed-length parts of packets, including any error detection and recovery that occurred in the physical layer.

- Network layer – provides connections and routes packets in the communication network, including handling the address of outgoing packets, decoding the address of incoming packets, and maintaining routing information for proper response to changing load levels.

- Transport layer – responsible for low-level network access and for message transfer between clients, including partitioning messages into packets, maintaining packet order, controlling flow, and generating physical addresses.
- Session layer – implements sessions, or process-to-process communications protocols.
- Presentation layer – resolves the differences in formats among the various sites in the network, including character conversions, and half duplex/full duplex (echoing).
- Application layer – interacts directly with the users; deals with file transfer, remote-login protocols and electronic mail, as well as schemas for distributed databases.