MODULE5

Mass-Storage Systems, File-System Interface and Implementation

Overview of Mass Storage Structure

- ∑ Magnetic disks provide bulk of secondary storage of modern computers
  - o Drives rotate at 60 to 250 times per second
  - o Transfer rate is rate at which data flow between drive and computer
  - o Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency)
  - o Head crash results from disk head making contact with the disk surface
    - –> That's bad
- ∑ Disks can be removable
- ∑ Drive attached to computer via I/O bus
  - o Busses vary, including EIDE, ATA, SATA, USB, Fibre Channel, SCSI, SAS, Firewire
  - o Host controller in computer uses bus to talk to disk controller built into drive or storage array

Magnetic Disks

- ∑ Platters range from .85" to 1->" (historically)
  - o Commonly 3.5", 2.5", and 1.8"
- ∑ Range from 30GB to 3TB per drive
- ∑ Performance
  - o Transfer Rate – theoretical – 6 Gb/sec
  - o Effective Transfer Rate – real – 1Gb/sec
  - o Seek time from 3ms to 12ms – 9ms common for desktop drives
  - o Average seek time measured or calculated based on 1/3 of tracks
  - o Latency based on spindle speed
    - –> 1/(RPM * 60)
  - o Average latency = ½ latency

Magnetic Tape

- ∑ Was early secondary-storage medium
  - o Evolved from open spools to cartridges
- ∑ Relatively permanent and holds large quantities of data
- ∑ Access time slow
- ∑ Random access ~1000 times slower than disk
- ∑ Mainly used for backup, storage of infrequently-used data, transfer medium between systems
- ∑ Kept in spool and wound or rewound past read-write head
- ∑ Once data under head, transfer rates comparable to disk
  - o 1->0MB/sec and greater
- ∑ 200GB to 1.5TB typical storage
- ∑ Common technologies are LTO-{3,->,5} and T10000

Disk Structure

- ∑ Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
- ∑ The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - o Sector 0 is the first sector of the first track on the outermost cylinder
  - o Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
  - o Logical to physical address should be easy

–> Except for bad sectors

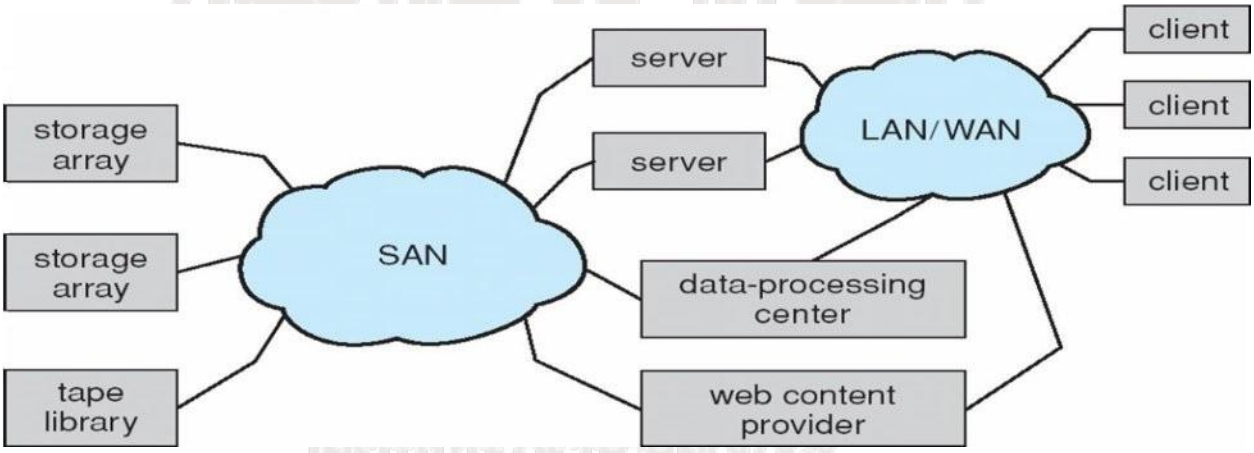
–> Non-constant # of sectors per track via constant angular velocity

Disk Attachment

- ∑ Host-attached storage accessed through I/O ports talking to I/O busses
- ∑ SCSI itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks
    - o Each target can have up to 8 **logical units** (disks attached to device controller)
    - o FC is high-speed serial architecture
    - o Can be switched fabric with 2->-bit address space – the basis of **storagearea networks (SANs)** in which many hosts attach to many storage units
- ∑ I/O directed to bus ID, device ID, logical unit (LUN)

Storage Area Network

- ∑ Common in large storage environments
- ∑ Multiple hosts attached to multiple storage arrays – flexible

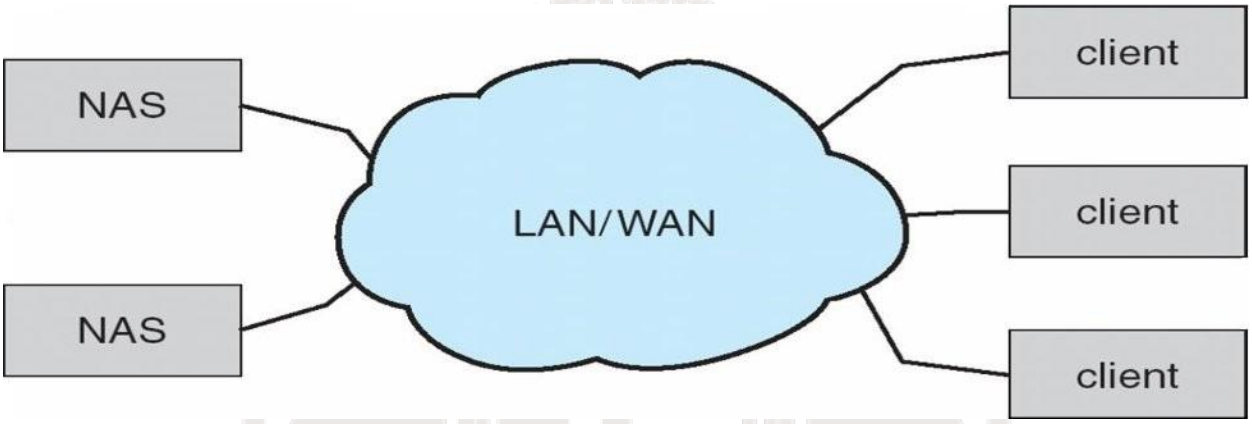


- ∑ SAN is one or more storage arrays
    - o Connected to one or more Fibre Channel switches
- ∑ Hosts also attach to the switches
- ∑ Storage made available via **LUN Masking** from specific arrays to specific servers
- ∑ Easy to add or remove storage, add new host and allocate it storage
    - o Over low-latency Fibre Channel fabric
    - o Why have separate storage networks and communications networks?

- Consider iSCSI, FCOE

Network-Attached Storage

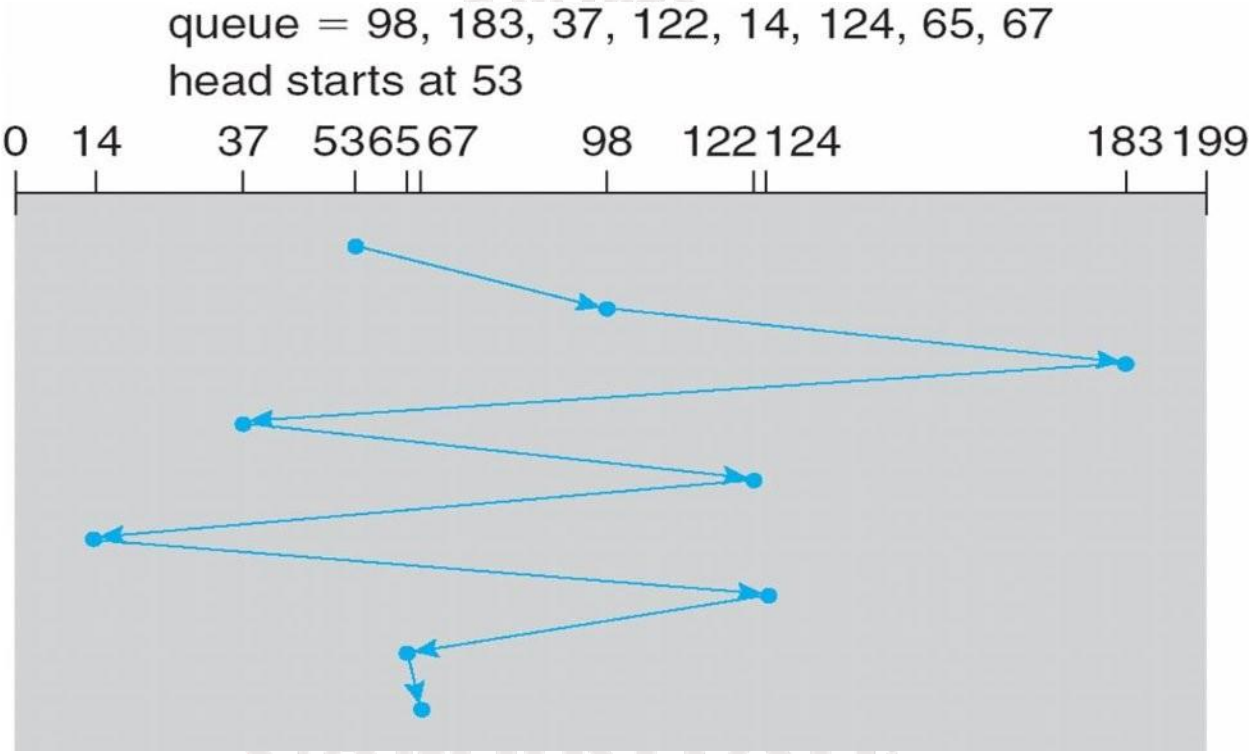
- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
    - Remotely attaching to file systems
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network
- **iSCSI** protocol uses IP network to carry the SCSI protocol
    - Remotely attaching to devices (blocks)



Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ♠ seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
- There are many sources of disk I/O request
- OS
- System processes
- Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue

- ∑ Optimization algorithms only make sense when a queue exists
- ∑ Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- ∑ Several algorithms exist to schedule the servicing of disk I/O requests
- ∑ The analysis is true for one or many platters
- ∑ We illustrate scheduling algorithms with a request queue (0-199)
- ∑ 98, 183, 37, 122, 1->, 12->, 65, 67
- ∑ Head pointer 53

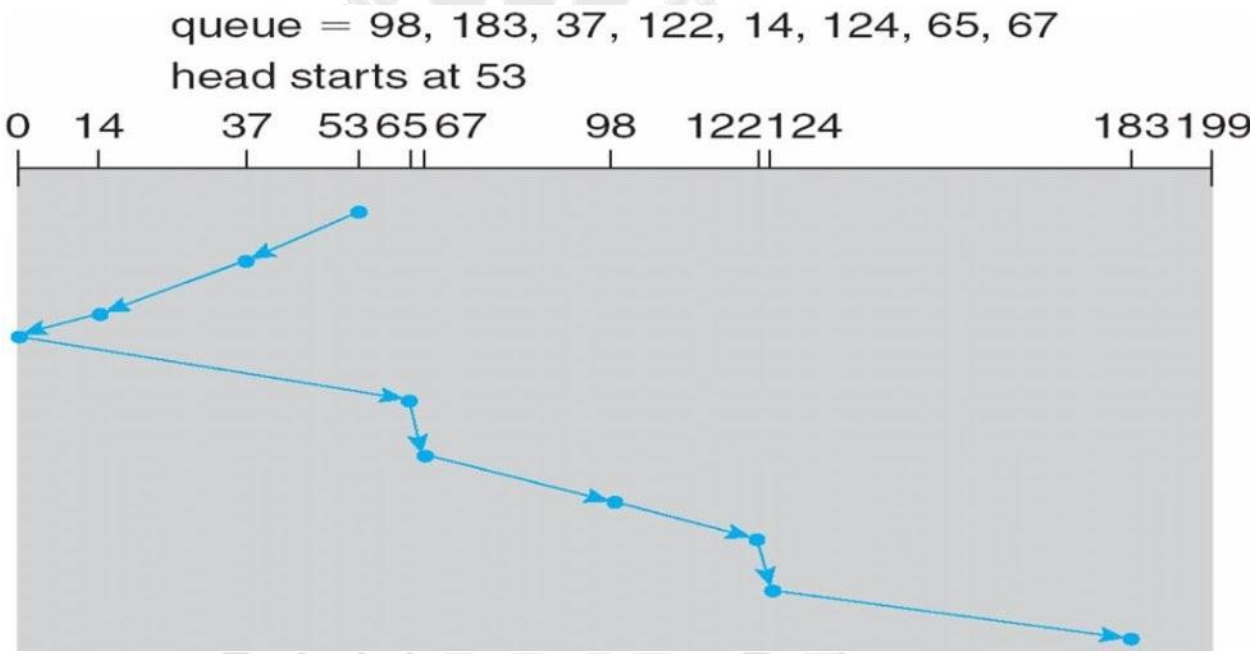


SSTF

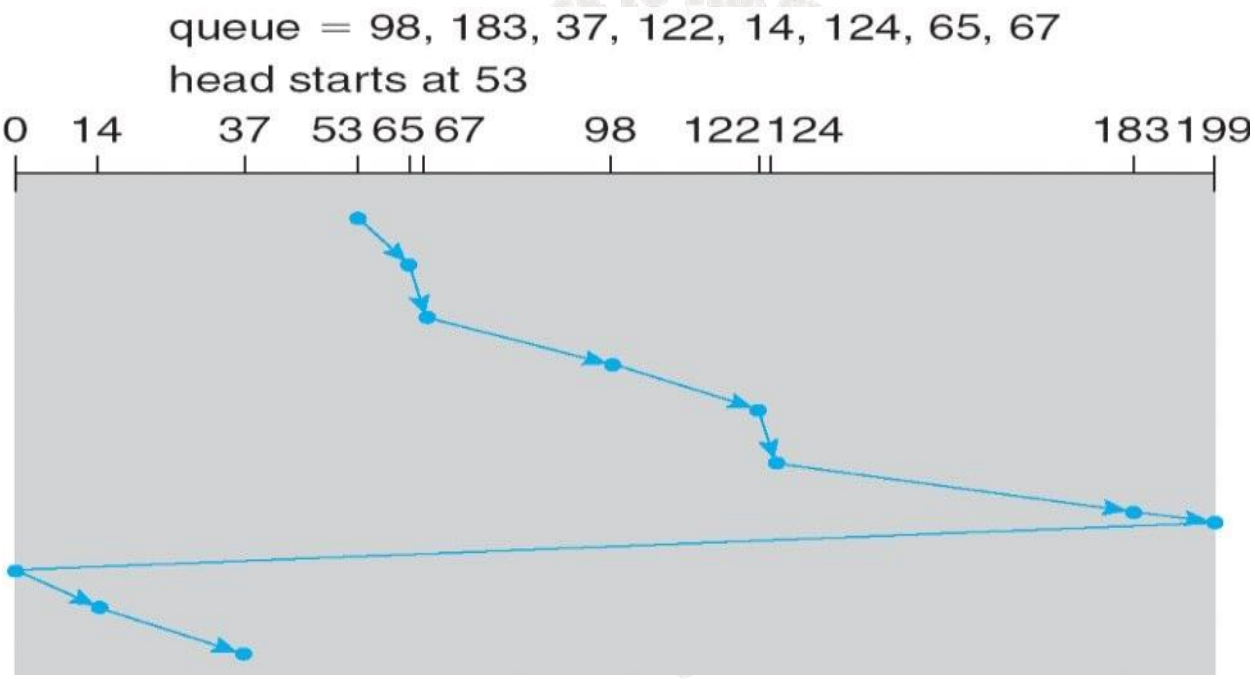
- ∑ Shortest Seek Time First selects the request with the minimum seek time from the current head position
- ∑ SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- ∑ Illustration shows total head movement of 236 cylinders

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Illustration shows total head movement of 208 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest



C-SCAN

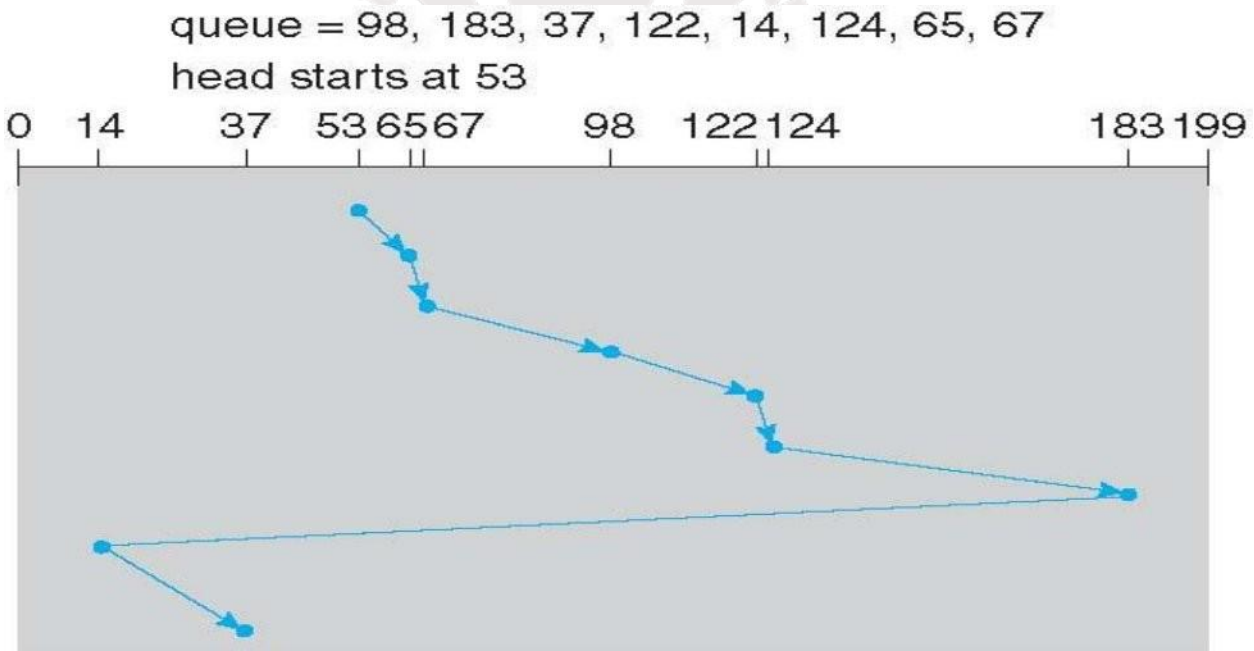


Provides a more uniform wait time than SCAN

- ∑ The head moves from one end of the disk to the other, servicing requests as it goes
  - o When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
  - o Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- ∑ Total number of cylinders?

C-LOOK

- ∑ LOOK a version of SCAN, C-LOOK a version of C-SCAN
- ∑ Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk



Disk Management

- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable
  - To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  - **Logical formatting** or "making a file system"
  - To increase efficiency most file systems group blocks into **clusters**
    - –> Disk I/O done in blocks
    - –> File I/O done in clusters
    - –> Boot block initializes system
  - The bootstrap is stored in ROM
  - **Bootstrap loader** program stored in boot blocks of boot partition
  - Methods such as **sector sparing** used to handle bad blocks

Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- Swap-space management
  - ->.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses **swap maps** to track swap-space use
  - Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created
    - –> File data written to swap space until write to file system requested
    - –> Other dirty pages go to swap space due to no other home
    - –> Text segment pages thrown out and reread from the file system as needed

RAID Structure

- RAID – multiple disk drives provides reliability via **redundancy**
- Increases the **mean time to failure**
- Frequently combined with **NVRAM** to improve write performance
- RAID is arranged into six different levels
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk **striping** uses a group of disks as one storage unit
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
  - **Mirroring** or **shadowing** (**RAID 1**) keeps duplicate of each disk
  - Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability
  - **Block interleaved parity** (**RAID ->, 5, 6**) uses much less redundancy
- RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common
- Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

Protection

Goals of Protection:

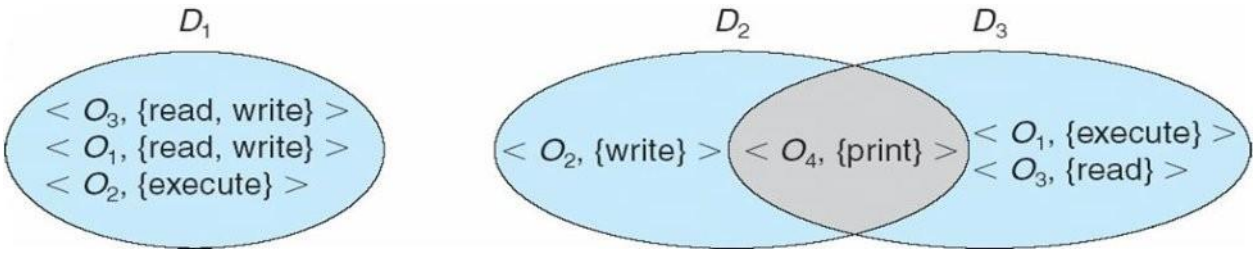
- In one protection model, computer consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed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

Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - Limits damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
  - "Need to know" a similar concept regarding access to data
- Must consider "grain" aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists, RBAC
    - Domain can be user, process, procedure

Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = set of access-rights

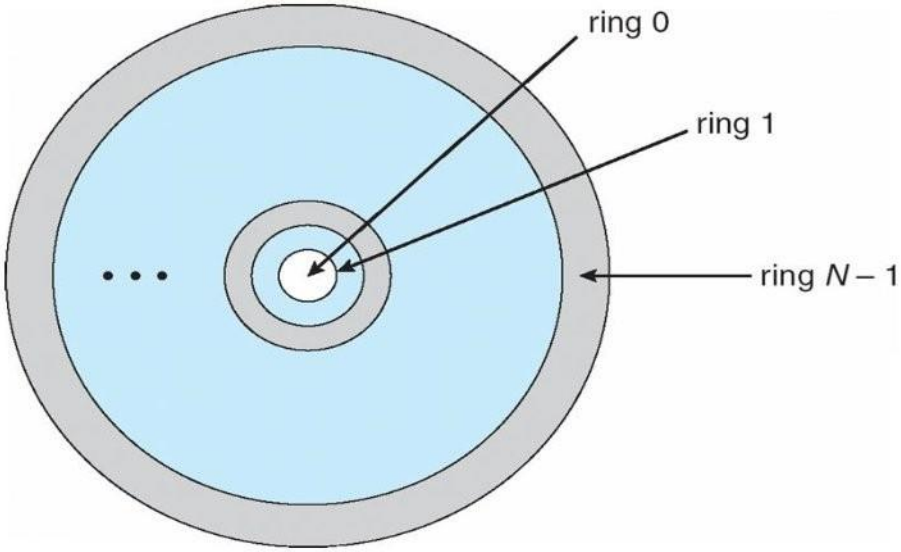


Domain Implementation (UNIX)

- 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
  - –> Domain switch accomplished via passwords
- su command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
- sudo command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain Implementation (MULTICS)

- Let D_i and D_j be any two domain rings
- If $j < I \Rightarrow D_i \prod D_j$



Access Matrix

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- *Access(i, j)* is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

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	

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
- User who creates object can define access column for that object
- 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 *(denoted by "*")* –> *control* – D_i *can modify* D_j

    access rights –> *transfer* – *switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object
- **Access matrix** design separates mechanism from policy
  - Mechanism

    –> Operating system provides access-matrix + rules

–> If 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

  –> But doesn't solve the general confinement problem

Access Matrix of Figure A with Domains as Objects

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

Access Matrix with *Copy* Rights

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix With *Owner* Rights

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Implementation of Access Matrix

- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples < *domain, object, rights-set* > in table
  - A requested operation M on object O_j within domain D_i -> search table for < D_i, O_j, R_k >

    -> with $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)
- Option 2 – Access lists for objects
  - Each column implemented as an access list for one object

- Resulting per-object list consists of ordered pairs < *domain, rights-set* > defining all domains with non-empty set of access rights for the object
- Easily extended to contain default set -> If M ∈ default set, also allow access

∑ 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 F1 – Read

Object F-> – Read, Write, Execute

Object F5 – Read, Write, Delete, Copy

∑ Option 3 – Capability list for domains

- Instead of object-based, list is domain based
- **Capability list** for domain is list of objects together with operations allows on them
- Object represented by its name or address, called a **capability**
- Execute operation M on object O_j, process requests operation and specifies capability as parameter

  –> Possession of capability means access is allowed
- Capability list associated with domain but never directly accessible by domain

  –> Rather, protected object, maintained by OS and accessed indirectly

  –> Like a "secure pointer"

  –> Idea can be extended up to applications

∑ Option -> – Lock-key

- Compromise between access lists and capability lists
- Each object has list of unique bit patterns, called **locks**

- Each domain as list of unique bit patterns called **keys**
- Process in a domain can only access object if domain has key that matches one of the locks

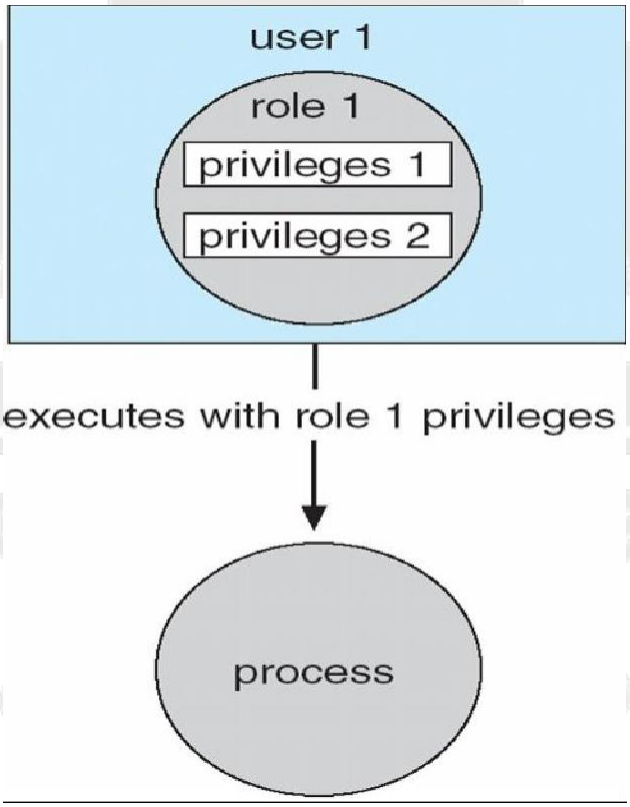
Access Control

∑ Protection can be applied to non-file resources

∑ Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege

- *Privilege* is right to execute system call or use an option within a system call
- Can be assigned to processes
- Users assigned *roles* granting access to privileges and programs

  –> Enable role via password to gain its privileges
- Similar to access matrix



Revocation of Access Rights

∑ Various options to remove the access right of a domain to an object

- Immediate vs. delayed
- Selective vs. general

- Partial vs. total
- Temporary vs. permanent

∑ **Access List** – Delete access rights from access list

- Simple – search access list and remove entry
- Immediate, general or selective, total or partial, permanent or temporary

∑ **Capability List** – Scheme required to locate capability in the system before capability can be revoked

- Reacquisition – periodic delete, with require and denial if revoked
- Back-pointers – set of pointers from each object to all capabilities of that object (Multics)
- Indirection – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
- Keys – unique bits associated with capability, generated when capability created

–> Master key associated with object, key matches master key for access

–> Revocation – create new master key

–> Policy decision of who can create and modify keys – object owner or others?

Capability-Based Systems

∑ Hydra

- Fixed set of access rights known to and interpreted by the system

–> i.e. read, write, or execute each memory segment

–> User can declare other **auxiliary rights** and register those with protection system

–> Accessing process must hold capability and know name of operation

-> **Rights amplification** allowed by trustworthy procedures for a specific type

- Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights
- Operations on objects defined procedurally – procedures are objects accessed indirectly by capabilities
- Solves the *problem of mutually suspicious subsystems*
- Includes library of prewritten security routines

- Cambridge CAP System
  - Simpler but powerful
  - **Data capability** - provides standard read, write, execute of individual storage segments associated with object – implemented in microcode
  - **Software capability** -interpretation left to the subsystem, through its protected procedures

    –> Only has access to its own subsystem

    –> Programmers must learn principles and techniques of protection

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

Security

The Security Problem:

- System **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- Intruders (crackers) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data