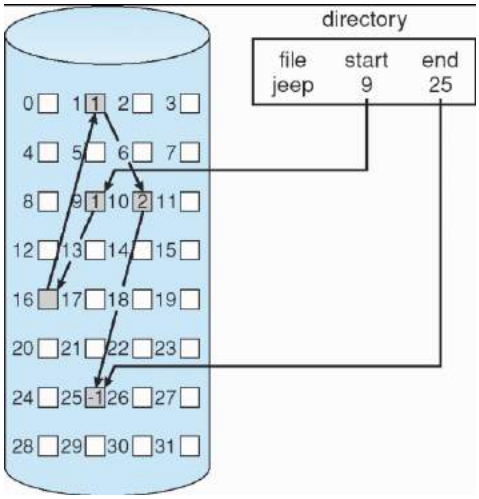
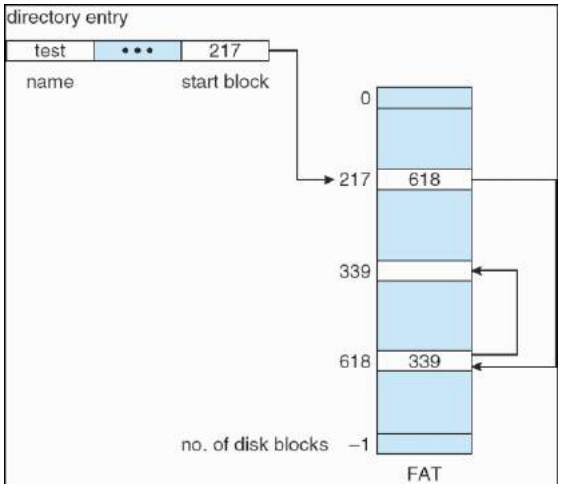
- To create a new file, just create a new entry in the directory
    - A write to the file causes a free block to be found
    - This new block is then written to and linked to the end of file
- **No external fragmentation, and any free block on the free-space list can be used to satisfy a request**
- The size of the file doesn't need to be declared on creation
- **Disadvantages**:
    - Can be used effectively only for **sequential-access** files
    - Space required for the **pointers**
        - Solution: collect blocks into multiples (‘**clusters’**) and allocate the clusters rather than blocks
    - **Reliability** (Problem if a pointer is lost / damaged)
- **Linked Allocation**:

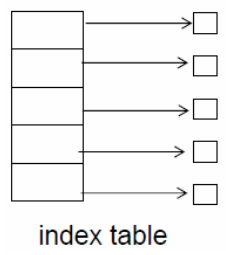


- **File Allocation Table (FAT)** = a variation on linked allocation:
    - A section of disk at the beginning of each partition is set aside to contain the table
    - The table has one entry for each disk block, and is indexed by block number
    - The **directory** entry contains the block number of the first block in the file
    - The table entry indexed by that block number then contains the block number of the next block in the file
    - This chain continues until the last block, which has a special end-of-file value as the table entry
    - The FAT can be cached to reduce the no. of disk head seeks
    - **Benefit**: improved access time, since the disk head can find the location of any block by reading the info in the FAT
- **File-Allocation Table**:

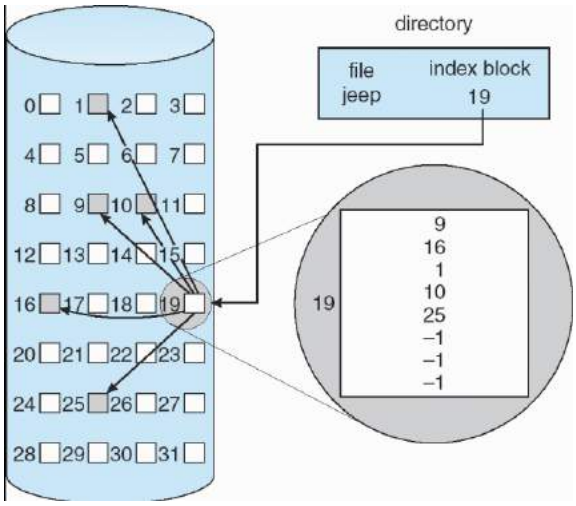


Indexed Allocation

- Solves the problems of linked allocation (without a FAT) by bringing all the pointers together into an index block
- Each file has an index block (an array of disk-block addresses)
- Logical view of the Index Table:

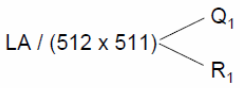


- The ith entry in the index block points to the ith file block
- The directory contains the address of the index block
- **Example of Index Allocation**:

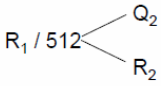


- When writing the ith block, a block is obtained from the free- space manager, and its address put in the ith index- block entry
- Supports **direct access**, without external fragmentation

- **Disadvantage**: **wasted space**: pointer overhead of the index block is greater than the pointer overhead of linked allocation
- We want the index block to be as small as possible, but what if file size is large?
    - Mechanisms to deal with the problem of index block size:
        - **Linked scheme**
            - To allow for large files, link several index blocks

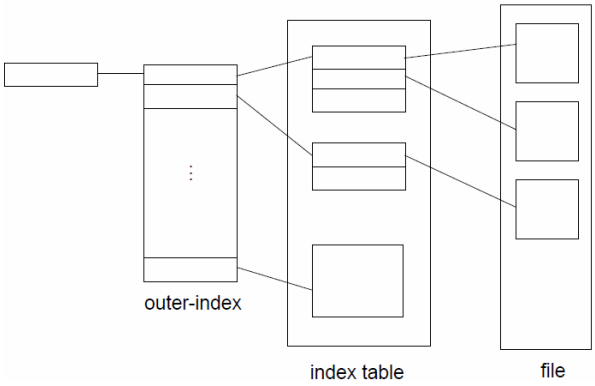


Q_1 = block of index table
R_1 is used as follows:

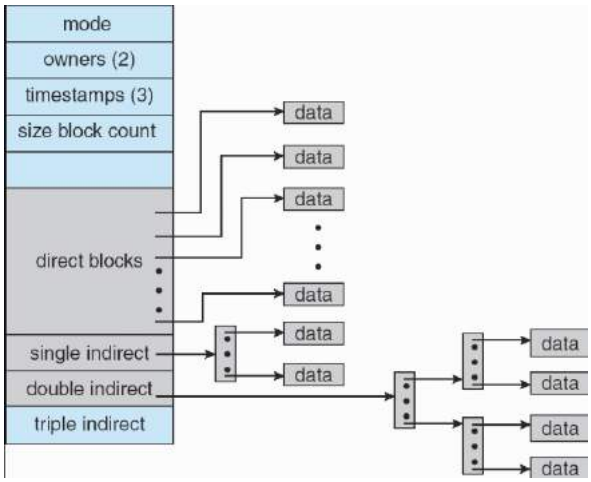


Q_2 = displacement into block of index table
R_2 displacement into block of file:

        - **Multilevel index**
            - A first-level index block points to second-level ones, which in turn point to the file blocks



        - **Combined scheme**
            - The first few pointers point to direct blocks
            - The next few point to indirect blocks
            - (Indirect blocks contain addresses of blocks)



- Indexed-allocation schemes suffer from some of the same performance problems as does linked allocation

Performance

- The above allocation methods vary in their storage efficiency and data-block access times
- Contiguous allocation
    - Requires only one access to get a disk block
    - Since we can keep the file's initial address in memory, we can calculate immediately the disk address of the ith block
    - Good for **direct access**
- Linked allocation
    - For direct access, you need i disk reads to access block i
    - Good for **sequential access**
- Indexed allocation
    - Performance depends on the index structure, the size of the file, and the position of the block desired

Free-Space Management

- A **free-space list** keeps track of free disk space
- To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file
    - When a file is deleted, its disk space is added to the list

Bit Vector

- The free-space list is implemented as a bit map / bit vector
- Each block is represented by a bit: 1 = free; 0 = allocated
- Advantage: relative simplicity & efficiency in finding the first free block, or n consecutive free blocks
- Disadvantage: Inefficient unless the entire vector is kept in main memory (and is written to disc occasionally for recovery)
- Also look in PDF and PTT notes

Linked List

- Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk
- The first block contains a pointer to the next free one, etc.
- Not efficient, because to traverse the list, each block is read
- Usually the OS simply needs a free block, and uses the first one

Grouping

- The addresses of n free blocks are stored in the 1st free block
- The first n-1 of these blocks are actually free
- The last block contains addresses of another n free blocks, etc

- Advantage: Addresses of a large no of free blocks can be found quickly, unlike in the standard linked-list approach

Counting

- Takes advantage of the fact that, generally, several contiguous blocks may be allocated / freed simultaneously
- Keep the address of the first free block and the number n of free contiguous blocks that follow the first block
- Each entry in the free-space list then consists of a disk address and a count

Space Maps

- p481 No description in notes...

Efficiency and Performance

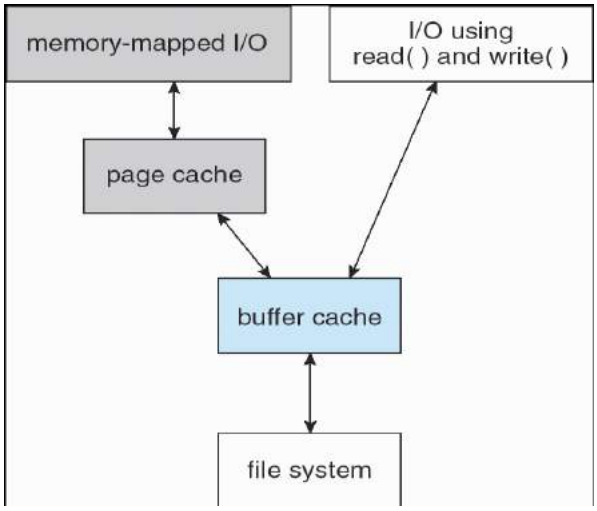
Efficiency

- The efficient use of disk space is heavily dependent on the disk allocation and directory algorithms in use
- The type of data kept in a directory also affect efficiency
    - Information like 'last write / access date' affect efficiency
- Small pointer sizes limit file length, but large ones take more space to store and make allocation methods use more disk space

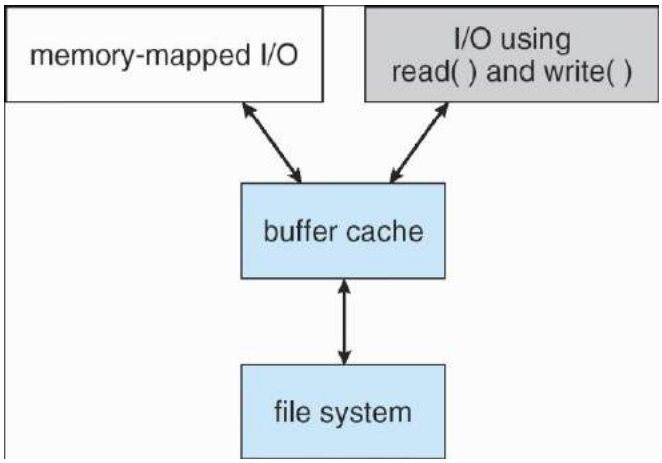
Performance

p483 - 486 for better description

- Most disk controllers include local memory to form an on-board **cache** that is large enough to store entire tracks at a time
- **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
- A **page cache** caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure:



- A unified buffer cache uses the same page cache to cache both memory- mapped pages and ordinary file system I/O



Recovery

- **System crashes** can cause inconsistencies among on-disk file-system data structures, such as directory structures, free-block pointers, and free FCB pointers.
    - Changes to these file-system structures can be performed in place and if interrupted by system crashes.
    - This causes the actual data on disk to be inconsistent to the state of the file-system structures.
- In addition to crashes, bugs in the file-system implementation, disk controllers, and even user applications can corrupt a file system.
- File systems have varying methods to deal with corruption, depending on the file-system data structures and algorithms.
- We deal with those issues next.

Consistency Checking

- Part of the directory info is kept in main memory (cache) to speed up access
- The directory info in main memory is more up to date than is the corresponding info on the disk
- When the computer crashes, the table of opened files is lost, as well as changes in the directories of opened files

- This event can leave the file system in an inconsistent state
- The consistency checker compares the data in the directory structure with the data blocks on disk and fixes inconsistencies (fsck in UNIX or chkdsk in Windows)
    - p487 top

Log-Structured File Systems

See also: Log-Based Recovery

- Similar idea to p260 section 6.9.2
- All metadata changes are written sequentially to a log
- Transaction = a set of operations that perform a specific task
- Once the changes are written to this log, they are committed
- When an entire committed transaction is completed, it is removed from the log file which is actually a circular buffer
- Side benefit of using logging on disk metadata updates: those updates proceed much faster than when they are applied directly to the on-disk data structures

Other Solutions

p488

Backup and Restore

- System programs can back up data from disk to permanent storage
- Use system programs to **back up** data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup
- To minimize copying, use info from each file's directory entry
- E.g. backup programs see when the last backup of a file was done
- Look at typical backup schedule on p.489 mid

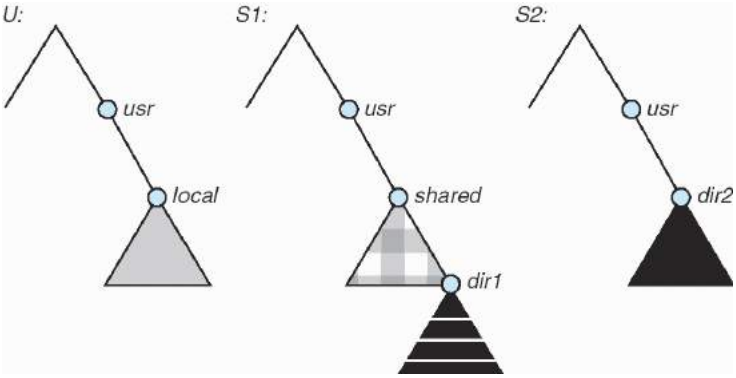
NFS

- NFS = An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet

Overview

- Sharing is based on a client-server relationship
- Sharing is allowed between any pair of machines
- To ensure machine independence, sharing of a remote file system affects only the client machine and no other machine
- For a remote directory to be accessible from a machine, a client of that machine has to carry out a **mount** operation first
- A remote directory is **mounted over** a directory of a local file system and looks like a sub-tree of the local file system
- The local directory becomes the name of the root of the newly mounted directory

- No transitivity: the client doesn't gain access to other file systems that were mounted over the former file system
- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
  - A remote directory is mounted over a local file system directory
    - The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
  - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
    - Files in the remote directory can then be accessed in a transparent manner
  - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory
- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services



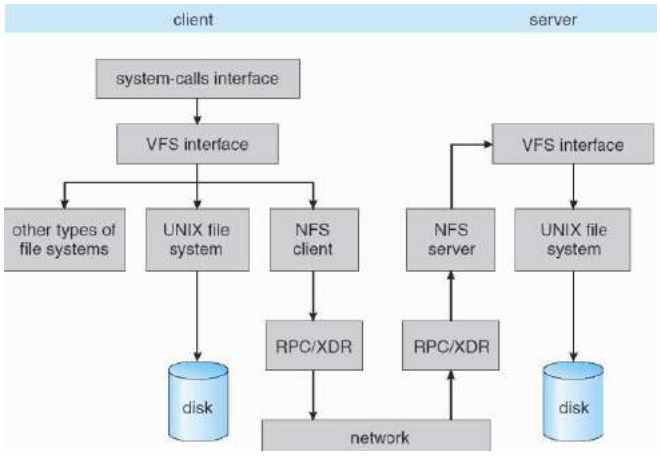
The Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list –specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
- File handle –a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side
- The server also maintains a list of clients and currently mounted directories, which it uses for admin purposes

The NFS Protocol

- Provides a set of RPCs for remote operations

- The procedures support the following operations:
    - Searching for a file within a directory
    - Reading a set of directory entries
    - Manipulating links and directories
    - Accessing file attributes
    - Reading & writing files
- These procedures can be invoked only after a file handle for the remotely mounted directory has been established
- NFS servers are **stateless** and don't maintain info about clients
- A server crash and recovery will be invisible to a client
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol doesn't provide concurrency-control mechanisms
- **Three Major Layers of NFS Architecture**
    - UNIX file-system interface (based on the **open, read, write,** and **close** calls, and **file descriptors**)
    - *Virtual File System*(VFS) layer –distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
        - The VFS activates file-system-specific operations to handle local requests according to their file-system types
        - Calls the NFS protocol procedures for remote requests
    - NFS service layer –bottom layer of the architecture
        - Implements the NFS protocol



Path-Name Translation

- Path-name translation is done by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
- Once a mount point is crossed, every component lookup causes a separate RPC to the server

Remote Operations

- A remote file operation can be translated directly to a RPC

Example: The WAFL File System

- Used on Network Appliance "Filers"–distributed file system appliances

- "Write-anywhere file layout"
- Serves up NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
- NVRAM for write caching
- Similar to Berkeley Fast File System, with extensive modifications

Summary

Mass-Storage Management

- Most programs are stored on disk until loaded into memory
- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long"period of time.
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
    - Free-space management
    - Storage allocation
    - Disk scheduling
- Some storage need not be fast
    - Tertiary storage includes optical storage, magnetic tape
    - Still must be managed
    - Varies between WORM (write-once, read-many-times) and RW (read-write)

Caching

- Cache management is important because of cache's limited size
- Data transfer from cache to CPU and registers is usually a hardware function, with no OS intervention
- Transfer of data from disk to memory is controlled by the OS
- Coherency and consistency
    - In the storage hierarchy, data appears in different levels
    - When it is modified, its value differs in the various storage
    - Its value only becomes the same throughout after being written from the internal register back to the e.g. magnetic disk
    - In a computing environment where one process executes at a time, access to the data will be to the copy at the highest level
    - In a multitasking environment, care must be taken to ensure that the several processes access the most recently updated value
    - In a multiprocessor environment, where the data exists in several caches simultaneously, an update to the data in one cache must be immediately reflected in all other caches where it resides (Cache coherency)
- In a distributed environment, when a replica is updated in one place, all other replicas must be brought up-to-date

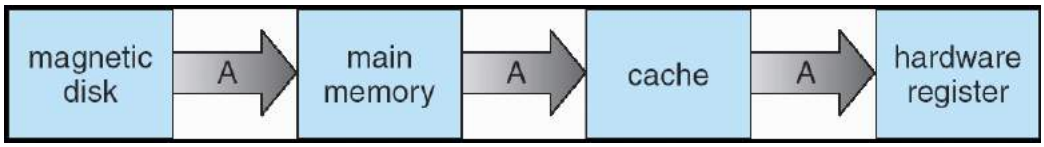
****

- Caching is an important principle of computer systems
    - Information is normally kept in some storage system (such as main memory)
    - As it is used, it is copied into a faster storage system - the cache - on a temporary basis
    - When we need a particular piece of information, we first check whether it is in the cache
        - If it is, we use the information directly from the cache
        - If it is not, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon
- In addition, internal programmable registers, such as index registers, provide high-speed cache for main memory
    - The programmer (or compiler) implements the register-allocation and register-replacement algorithms to decide which information to keep in registers and which to keep in main memory
    - There are also caches that are implemented totally in hardware
        - For instance, most systems have an instruction cache to hold the instructions expected to be executed next
    - Without this cache, the CPU would have to wait several cycles while an instruction was fetched from main memory
    - For similar reasons, most systems have one or more high-speed data caches in the memory hierarchy
    - We are not concerned with these hardware-only caches in this text, since they are outside the control of the operating system
- Because caches have limited size, **cache management** is an important design problem
    - Careful selection of the cache size and of a replacement policy can result in greatly increased performance
    - The figure below compares storage performance in large workstations and small servers

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

    - Various replacement algorithms for software-controlled caches are discussed in Chapter 9
- Main memory can be viewed as a fast cache for secondary storage, since data in secondary storage must be copied into main memory for use, and data must be in main memory before being moved to secondary storage for safekeeping
    - The file-system data, which resides on secondary storage, may appear on several levels in the storage hierarchy
        - At the highest level, the operating system may maintain a cache of file-system data in main memory

        - In addition, electronic RAM disks (also known as solid-state disks) may be used for high-speed storage that is accessed through the file-system interface
    - The bulk of secondary storage is on magnetic disks
        - The magnetic-disk storage, in turn, is often backed up onto magnetic tapes or removable disks to protect against data loss in case of hard-disk failure
        - Some systems automatically archive old file data from secondary storage to tertiary storage, such as tape jukeboxes, to lower the storage cost
- The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software
    - For instance, data transfer from cache to CPU and registers is usually a hardware function, with no operating system intervention
    - In contrast, transfer of data from disk to memory is usually controlled by the operating system
- In a hierarchical storage structure, the same data may appear in different levels of the storage system
    - For example, suppose that an integer A that is to be incremented by 1 is located in file B, and file B resides on magnetic disk
        - The increment operation proceeds by first issuing an I/O operation to copy the disk block on which A resides to main memory
        - This operation is followed by copying A to cache and to an internal register
        - Thus, the copy of A appears in several places:
            - on the magnetic disk
            - in main memory
            - in the cache
            - and in an internal register
            - (see the next figure)



            - Once the increment takes place in the internal register, the value of A differs in the various storage systems
            - The value of A becomes the same only after the new value of A is written from the internal register back to the magnetic disk
- In a computing environment where only one process executes at a time, this arrangement poses no difficulties, since an access to integer A will always be to the copy at the highest level of the hierarchy
    - However, in a multitasking environment, where the CPU is switched back and forth among various processes, extreme care must be taken to ensure that, if several processes wish to access A, then each of these processes will obtain the most recently updated value of A
- The situation becomes even more complicated in a multiprocessor environment where, in addition to maintaining internal registers, each of the CPUs also contains a local cache
    - In such an environment, a copy of A may exist simultaneously in several caches
    - Since the various CPUs can all execute concurrently, we must make sure that an update to the value A in one cache is immediately reflected in all other caches where A resides
    - This situation is called **cache coherency**, and it is usually a hardware problem (handled below the operating-system level)

- In a distributed environment, the situation becomes even more complex
    - In this environment, several copies (or replicas) of the same file can be kept on different computers that are distributed in space
    - Since the various replicas may be accessed and updated concurrently, some distributed systems ensure that, when a replica is updated in one place, all other replicas are brought up to date as soon as possible
    - There are various ways to do this - discussed in Chapter 17

I/O Systems

- The OS must hide peculiarities of hardware devices from users
- In UNIX, the peculiarities of I/O devices are hidden from the bulk of the OS itself by the I/O subsystem
- The I/O subsystem consists of
    - A memory-management component that includes buffering, caching, and spooling
    - A general device-driver interface
    - Drivers for specific hardware devices
- Only the device driver knows the peculiarities of the specific device to which it is assigned

Protection and Security

- **Protection**–any mechanism for controlling access of processes or users to resources defined by the OS
- **Security**–defense of the system against internal and external attacks
    - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
    - User identities (**user IDs**, security IDs) include name and associated number, one per user
    - User ID then associated with all files, processes of that user to determine access control
    - Group identifier (g**roup ID**) allows set of users to be defined and controls managed, then also associated with each process, file
    - **Privilege escalation** allows user to change to effective ID with more rights

PART SIX: PROTECTION AND SECURITY

Chapter 14: System Protection

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access
- Examine capability and language-based protection systems

Goals of 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.
- Reasons for providing protection:
    - To prevent mischievous violation of an access restriction

- To ensure that each program component active in a system uses system resources only in ways consistent with policies

- Mechanisms are distinct from policies
  - Mechanisms determine how something will be done
  - Policies decide what will be done
- This principle provides flexibility

Principles of Protection

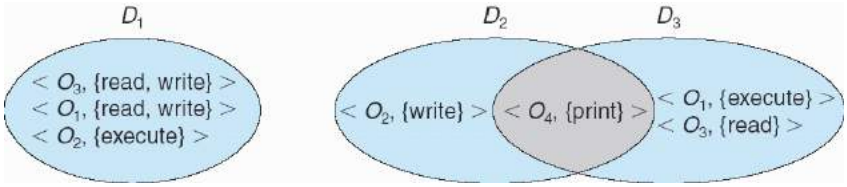
- Guiding principle –principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks
- p.592

Domain of Protection

- A process should be allowed to access only authorized resources
- Need-to-know principle: a process should be able to access only those resources that it currently requires to complete its task

Domain Structure

- A protection domain specifies the resources a process may access
- A domain is a collection of access rights, each of which is an ordered pair <object-name, rights-set>
  - Access right = the ability to execute an operation on an object
  - Access-right = <*object-name*, *rights-set*> where *rights-set* is a subset of all valid operations that can be performed on the object
  - Domains also define the types of operations that can be invoked



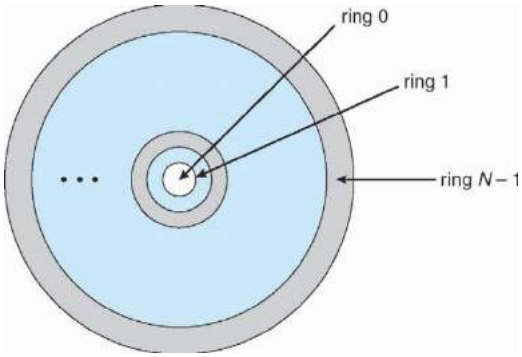
- The association between a process and a domain may be
  - Static (if the process' life-time resources are fixed)
    - Violates the need-to-know principle
  - Dynamic
    - A process can switch from one domain to another
- A domain can be realized in several ways:
  - Each user may be a domain
    - Domain switching occurs when a user logs out
  - Each process may be a domain
    - Domain switching occurs when a process sends a message to another process and waits for a response
  - Each procedure may be a domain
    - Domain switching occurs when a procedure call is made

An Example: UNIX

- 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 (set uid 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.

An Example: MULTICS

- Let *Di* and *Dj* be any two domain rings.
- If $j < i \Rightarrow Di \subseteq Dj$



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 Domaini can invoke on Objectj
- Example:

	F1	**F2**	**F3**	**Printer**
D1	Read		Read	
D2				Print
D3		Read	Execute	
D4	Read, Write		Read, Write	

- When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain)
- We can include domains in the matrix to control domain switching
- In the following table, D1 can only switch to D2:

	F1	**F2**	**F3**	**Printer**	**D1**	**D2**	**D3**	**D4**

D1	Read		Read			Switch		
D2				Print			Switch	Switch
D3		Read	Execute					
D4	Read, Write		Read, Write		Switch			

- Allowing controlled change to the contents of the access-matrix entries requires three additional operations:
    - **Copy**
        - * denotes the ability for one domain to copy the access right to another domain (row)
        - If the right is then removed from the original domain, it is called a **transfer**, not a copy
        - The * isn't copied as well
    - **Owner**
        - Allows the addition and removal of rights
        - **'Owner' in a column means that the process executing in that domain can add / delete rights in that column**
    - **Control**
        - The control right is applicable only to domain objects
        - 'Control' in access(D2,D4) means that a process executing in domain D2 can modify the row D4
- The problem of guaranteeing that no information initially held in an object can migrate outside of its execution environments is called the **confinement problem**

Implementation of Access Matrix

- Here we look at ways to implement the access matrix.

Global Table

- Contains a set of ordered triples: <domain, object, rights-set>
- Drawbacks:
    - The table is usually large and can't be kept in main memory
    - It is difficult to take advantage of groupings, e.g. if all may read an object, there must be an entry in each domain

Access Lists for Objects

- Each column in access list can be implemented as an access list for one object
- Each column = Access-control list for one object
    - Defines who can perform what operation.

      Domain 1 = Read, Write

      Domain 2 = Read

      Domain 3 = Read

      ⋮
- Resulting list: <domain, rights-set> for all non-empty columns

- Corresponds to users' needs: When you create an object, you can specify which domains may access it
- Determining the set of access rights for a domain is difficult

Capability Lists for Domains

- A **capability list** for a domain is a list of objects together with the operations allowed on those objects
  - An object is represented by its physical name or address, called its capability
  - Simple possession of the capability means that access is allowed
- Resulting list: Objects, with operations allowed on them
- Capabilities are useful for localizing info for a given process
- Capabilities are distinguished from other data in one of 2 ways:
  - Each object has a **tag** to denote its type as either a capability or as accessible data
  - The program's address space can be split into two parts:
    - One part contains data, accessible to the program
    - The other part contains the capability list, accessible only to the OS
- Each Row = Capability List (like a key)
  - Fore each domain, what operations allowed on what objects.

    Object 1 -Read

    Object 4 -Read, Write, Execute

    Object 5 -Read, Write, Delete, Copy

A Lock-Key Mechanism

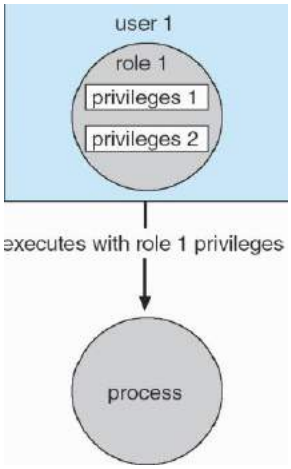
- Compromise between access lists and capability lists
- Each object has locks and each domain has keys
  - Unique bit patterns
- A process executing in a domain can access an object only if that domain has a key that matches one of the object's locks
- Not accessible by users directly

Comparison

- Check p.604

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



Revocation of Access Rights

- Questions about revocation that may arise:
  - Does it occur immediately or is it delayed?
  - Does it affect all users or only a group?
  - Can a subset of rights be revoked, or must they all be?
  - Can access be revoked permanently or temporarily?
- *Access List* - Delete access rights from access list.
  - Simple
    - Search the access list for the right(s) to be revoked, then delete it once found
  - Immediate
- Revocation is more difficult with capabilities:
  - Capabilities are distributed throughout the system, so we must find them first
  - *Capability List* - Scheme required to locate capability in the system before capability can be revoked:
    - Reacquisition
      - Periodically, capabilities are deleted from each domain, and the process may try to reacquire the capability
    - Back-pointers
      - Pointers point to all capabilities associated with an object, and can be followed when revocation is required
    - Indirection
      - Each capability points to an entry in a global table, which in turn points to the object
    - Keys
      - A key is associated with each capability and can't be modified / inspected by the process owning the capability
      - Master key is associated with each object; can be defined or replaced with the *set-key* operation

Capability-Based Systems

- These systems are not widely used.

An Example: Hydra

- Capability based system that provides flexibility
- Fixed set of access rights known to and interpreted by the system
    - Access rights:
        - read
        - write
        - execute a memory segment
- User can declare other rights.
- Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights.
- Operations on objects are defined procedurally.
    - These procedures are objects themselves and are accessed indirectly by capabilities.
    - When the definition of an object is made known by Hydra, the names of operations on the type become **auxiliary rights.**
- Auxiliary rights are described in a capability for an instance of the type.
- Provides **rights amplification.**
    - Procedures are certified as trustworthy to act on a formal parameter of a specified type on behalf of any process that holds a right to execute the procedure.
- Amplification allows implementation procedures access to the representation variables of an abstract data type.
- When a process invokes the operation *P* on an object *A*, however, the capability for access to *A* may be amplified as control passes to the code body of *P*.
- When a user passes an object as an argument to a procedure, we may need to ensure that the procedure cannot modify the object.
- The procedure-call mechanism of Hydra was designed as a direct solution to the problem of mutually suspicious subsystems.
- A Hydra subsystem is built on top of its protection kernel and may require protection of its own components.

An Example: Cambridge CAP System

- CAP's capability system is simpler and less powerful than that of Hydra.
- It also provide secure protection of user-defined objects.
- Two kinds of capabilities:
    - 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.
- The interpretation of a software capability is left completely to the subsystem, through the protected procedures it contains.

Language-Based Protection

- To the degree that protection is provided in existing computer systems, it is usually achieved through an operating-system kernel, which acts as a security agent to inspect and validate each attempt to access a protected resource.
- Protection systems are now concerned not only with the identity of a resource to which access is attempted but also with the functional nature of that access. In the newest protection systems, concern for the function to be invoked extends beyond a set of system-defined functions, such as standard file access methods, to include functions that may be user-defined as well.
- Protection can no longer be considered a matter of concern only to the designer of an operating system. It should also be available as a tool for use by the application designer, as that resources of an applications subsystem can be guarded against tampering or the influence of an error.

Compiler-Based Enforcement

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.
- When protection is declared along with data typing, the designer of each subsystem can specify its requirements for protection, as well as its need for use of other resources in a system.
    - Such specification should be given directly as a program is composed, and in the language in which the program itself is stated.
- This approach has several significant advantages:
    - Protection needs are simply declared, rather than programmed as a sequence of calls on procedures of an operating system.
    - Protection requirements can be stated independently of the facilities provided by a particular operating system.
    - The means for enforcement need not be provided by the designer of a subsystem.
    - A declarative notation is natural because access privileges are closely related to the linguistic concept of data type.
- A variety of techniques can be provided by programming-language implementation to enforce protection, but any of these must depend on some degree of support from an underlying machine and its operating system.
- If a system does not provide a protection kernel as powerful as those of Hydra or CAP, protection can still be implemented using specifications given in a programming language.
    - This kind of security will not be as good as protection implemented using a protection kernel. This is because the mechanism relay more on assumptions about the operational state of the system.
- The merits of security enforced by kernel opposed to enforcement by compiler:
    - Security
        - Enforcement by a kernel provides a greater degree of security of the protection system itself than does the generation of protection-checking code by a compiler.
    - Flexibility
        - There are limits to the flexibility of a protection kernel in implementing a user defined policy, although it may supply adequate facilities for the system to provide enforcement of its own policies
        - With a programming language, protection policy can be declared and enforcement provided as needed by an implementation.

    - Efficiency
        - The greatest efficiency is obtained when enforcement of protection is supported directly by hardware (microcode).
        - Language based enforcement has the advantage that static access enforcement can be verified off-line at compile time.
        - Intelligent compilers can tailor the enforcement mechanism so that the fixed overhead of kernel calls can often be avoided.
- Read the summary on p.612 (mid)
- What is needed is a safe, dynamic access-control mechanism for distribution capabilities to system resources among user processes.
- To be useful in practice it soul be reasonably efficient.
    - This has led to the development of a number of language constructs that allow the programmer to declare various restrictions on the use of specific managed resources.
    - These resources provide mechanisms for three functions:
        - Distributing capabilities safely and efficiently among customer processes.
        - Specifying the type of operations that a particular process may invoke on an allocated resource.
        - Specifying the order in which a particular process may invoke the various operations of a resource.
- 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.

Protection in Java

- Protection is handled by the Java Virtual Machine (JVM)
- A class is assigned a protection domain when it is loaded by the JVM.
- The protection domain indicates what operations the class can (and cannot) perform.
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library.
    - Look at example p.614 (Top)
- Stack inspection:
    - This philosophy require the class to explicitly permit a network connection.
    - By doing this the method takes responsibility for the request.
- The following figure shows stack inspection:

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	...
	get(url);
	open(addr);
	...
		get(URL u):
		...
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		...
			open(Addr a):
			...
			checkPermission (a, connect);
			connect (a);
			...

- A Java program cannot directly access memory; it can manipulate only an object for which it has a reference.
- Java's load-time and run-time checks enforce **type safety** of Java classes
    - Type safety ensures that classes cannot treat integers as pointers, write past the end of an array, or otherwise access memory in arbitrary ways.
- This is the foundation of Java protection, since it enables a class to effectively encapsulate and protect its data and methods from other classes loaded in the same JVM.

Summary

Chapter 15: System Security

- **Protection** (Ch 14) is strictly an internal problem: How controlled access to programs and data stored in a computer is provided.
- **Security** on the other hand, requires not only an adequate protection system but also consideration of the external environment within which the system operates.
- A protection system is effective if user authentication is compromised or a program is rum by an unauthorized user.
- Computer resources must be guarded against unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency.
- These resources include information stored in the system (code and data), as well as the CPU, memory, disks, tapes, and networking.
- Here we start by examining ways in which resources may be accidentally or purposely misused.
- We also look at key security enablers like cryptography.
- We then look at mechanisms to guard against or detect attacks.

Chapter Objectives:

- To discuss security threads and attacks
- To explain the fundamentals of encryption, authentication, and hashing
- To examine the uses of cryptography in computing
- To describe various countermeasures to security attacks

The Security Problem

- A system is **secure** if its resources are used and accessed as intended under all circumstances.
- Security violations (or misuse) of the system can be categorized as intentional (malicious) or accidental.
    - Easier to protect against accidental misuse than against malicious misuse.
- Note the following:
    - An **intruder** or **attacker** are people who are attempting to break security.
    - A **thread** is the potential for a security violation, whereas an **attack** is the attempt to break security.
- A list of several forms of accidental and malicious security violations (p.622):
    - Breach of **confidentiality**:
        - The unauthorized reading of data
    - Breach of **integrity**:
        - The unauthorized modification of data

- Breach of **availability**:
  - The unauthorized destruction of data (ex. defacement)
- **Theft of service**:
  - The unauthorized use of resources
- **Denial of service** (DOS):
  - The prevention of the legitimate use of a system

- Standard methods used to attempt to breach security:
  - **Masquerading**:
    - One participant in a communication pretends to be someone else
  - **Replay attack** (Captured exchange):
    - The malicious or fraudulent repeat of a valid data transmission
    - Used with message modification to escalate privileges
  - **Man-in-the-middle attack**:
    - The attacker sits in the data flow of a communication, masquerading as the sender to the receiver, and vice versa
    - Used after a session hijacking took place where the active communication session was intercepted

- To protect a system, we must take security measures at **4 levels**:

  1) Physical (Armed entry)

  2) Human (Users being bribed / tricked)

  - Social Engineering
  - Phishing
  - Dumpster diving

  3) Operating System (Security breaches to the system)

  4) Network (Intercepting data over private lines)

- Security at the first two levels must be maintained if operating-system security is to be secured.
- The system must provide protection (ch 14) to allow the implementation of security features.

Program Threats

- Writing a program to create a breach of security or causing a normal process to change its behavior and create a breach is a common goal of crackers
- While it is useful to log into a system without authorization it is even more useful to leave behind a back-door daemon that provides information or allows easy access even if the original exploit is blocked
- Multi-user computers are generally not prone to viruses because the executable programs are protected from writing by the OS

Trojan Horse

- A code segment that misuses its environment
- Examples:
  - login program emulation
  - Spyware (pop-ups, covert channels - where surreptitious communication occurs

Trap Door

- A programmer leaves a hole in the software that only he can use

Logic Bomb

- A program that initiates a security incident only under certain circumstances (e.g. date and time or looking for a specific parameter to change)

Stack and Buffer Overflow

- The most common way for an attacker to gain unauthorized access
- An authorized user may also use this exploit for privilege escalation, to gain privileges beyond those allowed for him
- Essentially, the attack exploits a bug in a program
- The attacker finds the vulnerability and writes a program to:
    - Overflow an input field, command-line argument, or input buffer… until it writes into the stack
    - Overwrite the current return address on the stack with the address of the exploit code loaded in step 3
    - Write a simple set of code for the next space in the stack that includes commands that the attacker wants to execute
- This attack can be countered by doing bounds checking on inputs
- **Read p.628-629 NB!!!**

Viruses

- A virus is a fragment of code embedded in a legitimate program
- Whereas a worm is structured as a complete, standalone program, a virus is a fragment of code embedded in a legitimate program
- Viruses are very specific to architectures
- Viruses are normally hidden in Trojan horse programs acting as virus droppers
- Viruses normally belong to more than one category:
    - **File**:
        - A virus appends itself to a file then after the program is executed it removes itself from the file and return control to the program
    - **Boot:**
        - It infects the boot sector and executes every time the computer boots. It sits in memory and infects all other boot sector disks which is inserted to the computer
    - **Macro:**
        - Written in a high-level language and triggered when a program capable of executing the macro is run
    - **Source code:**
        - This virus looks for source code and include the virus to help distribute the virus
    - **Polymorphic:**
        - Virus changes each time installed to avoid detection. Done to change virus's signature. Signature is a pattern that is used to detect the virus
    - **Encrypted:**
        - Encrypted to avoid detection. Decryption included to decrypt and to infect a target
    - **Stealth:**
        - It attempts to avoid detection by modifying parts of the system that normally detects it

    - **Tunneling:**
        - Bypass detection by anti-virus programs by installing itself in interrupt-handler chain and into device drivers
    - **Multipartite:**
        - Infect multiple parts of the system (Boot sector; memory; files)
    - **Armored:**
        - Compressed to avoid detection and disinfection by anti-virus programs. Difficult to understand by anti- virus researchers
- **Keystroke logger**:
    - Record all thing entered on keyboard
- **Monoculture**:
    - An environment where many systems run the same hardware, operating system, and/or application software

System and Network Threats

- System and network threads involve the abuse of services and network connections
- System and network threats create a situation in which operating-system resources and user files are misused
- System and network attacks are used to launch a program attack, and vice versa
- The more **open** an operating system is the more services are running and the more likely a bug is available to be exploit
- The **attack surface** is the set of ways an attacker can try to break into a system

Worms

- **(p.634)**
- A worm spawns copies of itself, using up system resources
- The worm spawns copies of itself using up all system resources and locking out all other processes

Port Scanning

- Port scanning is a means for a cracker to detect a system's vulnerabilities to attack
- It normally attempts to create a TCP/IP connection to a specific port or a range of ports
- Port scans are normally launched from **zombie systems**
- Such systems are previously compromised systems that are serving their owners while being used for nefarious purposes, including denial-of-service attacks and spam relay

Denial of Service

- Involves disabling legitimate use of a system / facility
- E.g. an intruder could delete all the files on a system
- Generally network based attacks, and fall into two categories:
    - An attack that uses so many facility resources that no useful work can be done
    - Disrupting the network of the facility
- Distributed Denial-Of-Service attacks (DDOS):
    - Attacks launched from multiple sites at once

Cryptography as a Security Tool

- We look at details of cryptography and its use in computer security
- All computers on a network sent and receive bits onto and from the wire without knowing from where they come or where they go to
- Constrains the potential senders and receivers of a message
- Keys are distributed to computers to process messages
- Cryptography enables a recipient of a message to verify that the message was created by some computer possessing a certain key - the key is the source of the of the message

Encryption

- A means for constraining the possible **receivers** of a message
- An encryption algorithm enables the sender of a message to ensure that only a computer possessing a certain key can read the message
- An encryption algorithm consists of the following components:
    - A set K of keys
    - A set M of messages
    - A set C of ciphertexts
    - A function E:K->(M->C)
    - A function D:K->(C->M)
- An encryption algorithm must provide this essential property:
    - Given a chipertext cEC, a computer can compute m such that E(k)(m)=c only if it possesses D(k)
- Two main types of encryption algorithms:
    - Symmetric encryption algorithm
    - Asymmetric encryption algorithm

Symmetric Encryption

- In a symmetric encryption algorithm, the same key is used to encrypt and decrypt
- E(k) is derived from D(k), hence E(k) must be protected to the same extend as D(k)
- Data-Encryption Standard (DES)
    - Breaks messages up into 64-bit blocks
    - Keys are 56-bit key
    - Called a block cipher
    - Cipher-block chaining:
        - The chunks are XORed with the previous ciphertext
        - This is done to prevent that the ciphertext can be used to determine the encryption and decryption keys
    - Triple-DES was developed since DES was deemed insecure
- Advanced Encryption Standard (AES)
    - Breaks messages up into 128-bit blocks
    - Key lengths are 128, 192, 256-bits
- Stream ciphers are designed to encrypt and decrypt a stream of bytes or bits rather than a block

- RC4
- Use pseudo-random-bit generator to produce random bits which is fed a key and delivers a keystream
- A keystream is an infinite set of keys that can be used for the input plaintext stream
- RC4 is used to encrypt streams of data, such as WEP
- Used to encrypt Web communications between Web browsers and Web servers

Asymmetric Encryption

- In an asymmetric encryption algorithm, different keys are used to encrypt and decrypt
- RSA
  - A block-cipher public-key algorithm
  - The encryption key is called the public key and is distributed freely
  - The decryption key needs to be kept secret and is thus called the private key
- It is not feasible to calculate the decryption key from the encrypted text
- More safe and are used authentication, confidentiality and key distribution
- Needs more computer power to execute

Authentication

- Authentication is used to constrain the set of potential **senders** of a message
- Authentication is complementary to encryption
- Used to prove that a message has not been modified
- Components of an authentication algorithm:
  - A set K of keys
  - A set M of messages
  - A set A of authenticators
  - A function S:K->(M->A)
  - A function V:K->(MxA->{true, false}) (Verification function)
- A **hash function** H(m) creates a small fixed-size block of data, known as a **message digest** or **hash value**, from a message m
  - A hash function work by taking a message in n-bit blocks and processing the blocks to produce an n-bit hash
  - H must be collision resistant on m - that is, it must be infeasible to find an m' not equal to m such that H(m)=H(m')
  - If H(m)=H(m') we know that m=m' - that is, we know that the message has not been modified
  - Two common message-digest functions:
    - MD5 outputs 128-hash
    - SHA-1 outputs 160-bit hash
  - Message digests are useful for detecting changed messages but are not useful as authenticators
  - An authentication algorithm takes the message digest and encrypts it
- Two varieties of authentication algorithms:
  - **Message authentication code (MAC)**

        - A cryptographic checksum is generated from the message using a secret key
        - Knowledge of V(k) and knowledge of S(k) are equivalent: one can be derived from the other, so k must be kept secret
        - Because of the collision resistance in the hash function, we are reasonably assured that no other message could create the same MAC
    - **Digital signature algorithm (with public & private keys)**
        - The authenticators produced are called digital signatures
        - In a digital-signature algorithm, it is computationally infeasible to derive S(ks) from V(kv); in particular, V is a one-way function
        - kv is the public key and ks is the private key
        - RSA digital algorithm is one example
- The primary three reasons why we need encryption and authentication algorithms:
    - Authentication algorithms generally require fewer computations
    - The authenticator of the message is almost always shorter than the message and its cipher
    - Sometimes, we want authentication but not confidentiality
- Authentication is the core of nonrepudiation, which supplies proof that an entity performed an action
    - Ex: The filling out of an electronic form

Key Distribution

- Delivery of a symmetric key is a huge challenge
    - Done out-of-band: on paper or conversation
- Asymmetric keys can be exchanged in public and each the user needs only one private key
- To make sure that the public key is legit authentication takes place on the public key using a digital certificate
    - A **digital certificate** is a public key digitally signed by a trusted party
    - The trusted party receives proof of identification from some entity and certifies that the public key belongs to that party
    - The **certificate authorities** have their public keys included within the Web browsers before they are distributed, hence we know it's legit
- The digital certificates are distributed in the **standard X.509 digital certificate format**

Implementation of Cryptography

- Implementation of cryptography can happen at almost any one of the 7 seven OSI Model layers
- In general more protocols benefit from protections placed lower in the protocol stack (not definitive)
    - This might lead to lower protection in higher-layers of the OSI model
- IPSec (IP security) is used as the basis for VPNs
- p.646 - p.647

An Example: SSL

- Protocol that enables two computers to communicate securely
    - To limit the sender and receiver of messages to the other
- With SSL a client and server establishes a secure session key that can be used for symmetric encryption of the session between the two to avoid man-in-the-middle attacks

- Once the session is complete the session key is thrown away
- The SSL protocol is initiated by a **client c** to communicate securely with a **server s**
- The server obtain a certificate (cert) from certification authority **CA**
- The certificate contain the following:
    - Attributes of the server
    - Identity of a public encryption algorithm
    - Public key of this server
    - Validity interval of the certificate
    - Digital signature on the above information made by CA
- The client obtain the public verification algorithm prior to the protocol's use

User Authentication

- Protection depends on the ability to identify the programs and processes currently executing, which then must also be able to identify each user of the system
- How does the system determine if a user is authentic?
    - The user's possession of something (e.g. key or card)
    - The user's knowledge of something (e.g. identifier or password)
    - An attribute of the user (e.g. fingerprint or signature)

Passwords

- Used to protect access to a system together with a username
- Used to protect objects in the system (files etc.)
- Used to determine access rights

Password Vulnerabilities

- **Guessing** – through knowing the user / using brute force
    - Use good (longer) passwords to prevent guessing
- **Shoulder surfing** = looking over the shoulder of a user
    - Make sure people are not watching while entering password
- **Sniffing** = watching all data being transferred on the network
    - Encryption solves sniffing problem
- **Illegal transfer** – when users break account-sharing rules
- Passwords can be system-generated
- Some systems age passwords, forcing them to change at intervals

Encrypted Passwords

- Only encoded passwords are stored
- When a user presents a password, it is encoded and compared
- Thus, the password file doesn't need to be kept secret

One-Time Passwords

- **Different** password every time, so interceptors can't re-use it

- Paired passwords: the system presents one part and the user must supply the other part – the user is challenged
- System & user share a secret, which must not be exposed
- A seed is a random number / numeric sequence
    - The seed is the authentication challenge from the computer
- The secret and seed are input to a function: f(secret,seed)
- The result of the function is transmitted as the password
- Two-factor authentication: using a one-time password + PIN (personal identification number)
- Code book / one-time pad = a list of single-use passwords, which are crossed out after being used

Biometrics

- Biometrics used to secure physical access
    - ex. access to a data center
- Palm / finger-print readers can scan you
- **Multifactor authentication** is the use of various authentication methods to authenticate a user (password, fingerprint, etc...)

Implementing Security Defenses

- Includes improved user education, technology and writing bug-free software (not all)
- **Defense in depth** is a theory where people believe it is better to have more layers of defense than fewer layers

Security Policy

- First step to security is to have a security policy
- Statement of what is secured
- Without policy, users and admins wouldn't know what is permissible, required and what is not allowed
- Security policy document is a living document that is reviewed and updated periodically

Vulnerability Assessment

- To determine if security policy is correctly implemented a **vulnerability assessment** is needed
- This can consist out of broad range of tests, some of which are:
    - Social engineering
    - Risk assessment
    - Port scans
- Risk Assessments:
    - Look at the odds a security incident will affect the entity and decrease its value
- **Core activity on vulnerability assessments is penetration test**:
    - Scan the entity for known vulnerabilities
    - Done when computer use is relatively low
    - Do on test systems
    - Scans can check variety of aspects:
        - Short or easy-to-guess passwords

        - Unauthorized privileged programs (setuid)
        - Unauthorized programs in system directories
        - Unexpectedly long-running processes
        - Improper directory protections on user and system directories
        - Improper protections on system data files (password file, device drivers, OS kernel)
        - Dangerous entries in program search path (section 15.2.1)
        - Changes to system programs detected with checksum values
        - Unexpected or hidden network daemons
- A system is only as secure as its most far-reaching connection
    - If a system has a connection outside a building the system is not secure
- Vulnerability scans are used on networks to find problems with network security
    - Scans search the network for ports that responds to a request
    - Access to unsafe open ports can be blocked
    - Scans determine details of applications on ports by listening and try to find vulnerabilities for the application
    - Maybe system needs patches or is misconfigured
- Tools that are used to test security can be dangerous in the hands of a bad person
- **Security through obscurity**:
    - People advocate that no tools should be developed to test security, since it is used to find security holes

Intrusion Detection

- Encompasses many techniques that vary on a number of axes:
    - The time that detection occurs
    - The types of inputs examined to detect intrusive activity
    - The range of response capabilities
- Intrusion-Detection Systems (**IDSs**)
    - Raises an alarm when intrusion is detected
- Intrusion-Prevention Systems(**IDPs**)
    - Acts as routers, passing traffic unless an intrusion is detected
- What constitutes an intrusion?
    - **Signature-based detection**
        - System input is examined for specific behavior patterns
        - E.g. monitoring for multiple failed attempts to log on
        - 'Characterizes dangerous behavior and detects it'
    - **Anomaly detection**
        - E.g. monitoring system calls of a daemon process to detect if its system-call behavior deviates from normal patterns
        - 'Characterizes non-dangerous behavior and detects the opposite'
        - **Tripwire (Example of Anomaly-Detection Tool)**

- Operates on the premise that a large class of intrusions result in anomalous **modification** of system directories & files
- Tripwire is a tool to monitor file systems for added, deleted, or changed files, and to alert admin to these modifications
- **Limitations**:
  - The need to protect the Tripwire program
  - Some security-relevant files are *supposed to* change in time

Virus Protection

Auditing, Accounting, and Logging

- Audit-trail processing: security-relevant events are logged to an audit trail and then matched against attack signatures or analyzed for anomalous behavior

Firewalling to Protect Systems and Networks

- A firewall limits network access between two security domains
- It monitors and logs connections and can also limit connections
- A firewall can separate a network into multiple domains
- A Firewall don't prevent attacks that tunnel, or travel within protocols / connections that the firewall allows
- Another firewall vulnerability spoofing, where an unauthorized host pretends to be an authorized one by meeting some criteria

Computer-Security Classification

- Trusted Computer Base (TCB) = the sum total of all protection systems within a computer system that enforce a security policy

An Example: Windows XP

Summary

Distributed Systems

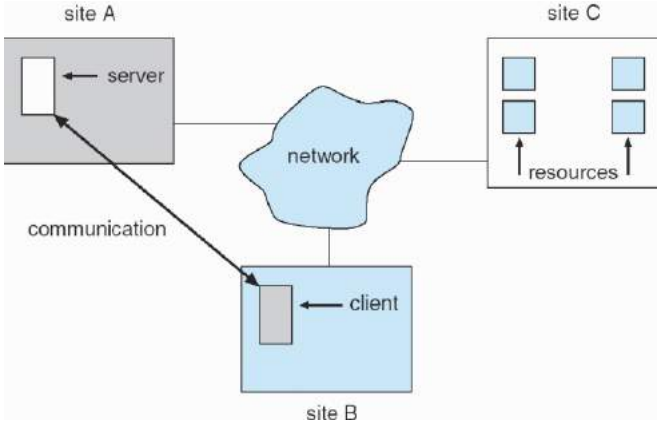
PART SEVEN: DISTRIBUTED SYSTEMS

Chapter 16: Distributed Operating Systems

- A distributed system is a collection of processors that do not share memory or a clock
- Instead, each processor has its own local memory
- The processors communicate with one another through various communication networks, such as high-speed buses or telephone lines
- Here we discuss the general structure of distributed systems and the networks that interconnect them
- We contrast the main differences in operating-system design between these systems and centralized systems
- **Chapter Objectives**:
  - To provide a high-level overview of distributed systems and the networks that interconnect them
  - To discuss the general structure of distributed operating systems

Motivation

- Distributed system is a collection of loosely coupled processors interconnected by a communication network
- Each processor has its own local resources
- The processors communicate through networks
- General structure of distributed system



- Four reasons for building Distributed Systems:
  - resource sharing
  - computation speedup
  - reliability
  - communication

Resource Sharing

- A user at one site can use the resources at another site
- E.g. sharing files, processing info, printing files…

Computation Speedup

- If a computation can be partitioned into sub-computations that can run concurrently, then a distributed system allows us to distribute the sub-computations among various sites
- If a particular site is overloaded with jobs, some may be moved to other lightly loaded sites (= load sharing)

Reliability

- If one site fails, the remaining ones can still operate
- If each machine is responsible for some crucial system function, then a single failure may halt the whole system
- With enough redundancy, the system can continue operation
- The failure of a site must be detected by the system
- The system must no longer use the services of that site
- If the function of the failed site can be taken over by another site, the system must ensure that the transfer of function occurs correctly
- When the failed site recovers / is repaired, mechanisms must be available to integrate it back into the system

Communication

- Messages passed between systems in the same way a single-computer message system does (section 3.4)
- Functions include file transfer, login, mail, and remote procedure calls (RPCs)
- These functions can be carried out over distances
- Users at different sites can exchange information
- Advantages:
    - Collaboration over distances
    - Downsizing

Types of Network-based Operating Systems

- Two general categories of network-oriented operating systems:
    - Network Operating Systems
    - Distributed Operating Systems

Network Operating Systems

- Simpler to implement, but more difficult for users to access and utilize than distributed OSs
- Provides an environment where users who are aware of multiplicity of machines can access remote resources on each others machines

Remote Login

- The Internet provides the telnet facility for this purpose
    - ex: telnet cs.yale.edu
    - Creates socket connection between the local machine and the cs.yale.edu computer
    - Open bidirectional connection
    - User must enter username and password
    - User can execute any command on remote computer as any local user can

Remote File Transfer

- Each computer maintains its own local file system
- User need access, else use anonymous and arbitrary password
    - Anonymous users can only access files in the directory with public access
    - Care must be taken that anonymous users cannot access files outside this directory
- The Internet provides the FTP program
    - ex. connect with: ftp cs.yal.edu
    - copy the file with: get Server.java
- This does not provide real file sharing
- User must know where the files are in the subdirectories
- Various copies of the same file can exist and they can be inconsistent
- Only predefined set of file-related commands can be used:
    - get - from remote to local machine
    - put - from local to remote machine
    - ls or dir - list files in current directory of remote machine

    - cd - change current directory of remote machine
- A windows user logging into a Unix machine using telnet should change paradigms
    - User must use Unix commands
- Distributed operating systems address this issue

Distributed Operating Systems

- Provide more features than network OSs
- Users access remote resources in the same manner as local ones
- Data and process migration is under control of distributed operating system

Data Migration

- If a user need to work on a remote file:
    - Transfer the entire file to the other site and back after modifying the file
        - Very inefficient if only small part of large file is being processed
        - Efficient if large portions of file needs to be processed
    - Transfer only the necessary portions of the file needed for immediate task
        - If other portion needed, can be transferred
        - After modification only the modified portion is transferred back to remote site

Computation Migration

- Invoke a procedure at another site and get the result
- Access to file carried out remotely and is initiated by RPC
    - RPC uses datagram protocol (UDP on Internet) to execute routine on remote site (section 3.6.2)
    - Can also send message to remote site
        - Remote site opens up new process, executes task, sent result back and close
        - Can be executed concurrently/bidirectional between sites

Process Migration

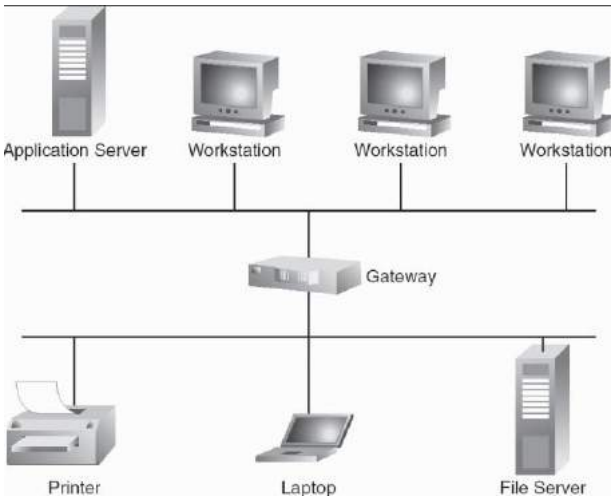
- Logical extension of computation migration
- Why (parts of) a process may be executed at different sites
    - Load balancing - even workload
    - Computation speedup - reduce total process turnaround time
    - Hardware preference - specialized processor/hardware needs
    - Software preference - software only available at specific site and not cost effective to move
    - Data access - if to much data must be moved, remote processes can be more efficient
- Two techniques to move processes in a network
    - The system can hide the fact that the process has migrated from the client
        - No user programming needed to accomplish process migration
    - The user must specify explicitly how the process should migrate
        - Done to satisfy a hardware or software preference

Network Structure

- Two type of networks:
    - Local Area Networks (LAN)
    - Wide Area Networks (WAN)
- Main difference is in the way they are geographically distributed
    - Local Area Networks (LAN)
        - Distributed over small areas (inside single or adjacent buildings)
    - Wide Area Networks (WAN)
        - Distributed over large areas (in a country)
- Differences in speed and reliability of communications networks

Local-Area Networks

- LANs emerged as a substitute for large mainframe computers
    - A number of small computers are used to replace the mainframe computers
    - Small computers have self-contained applications
    - Can be used for data sharing in the enterprise
- Designed to cover small geographical area
- Multiaccess bus, ring, or star network
- Because computers closer together, higher speed and lower error rate than computers in WAN
- Make use of twisted-pair and fiber-optic cabling
- Speeds of 10Mbps to 100Mbps can be obtained with the Ethernet protocol used for LANs
- Normally consists out of computers, shared peripheral devices and one or more gateways

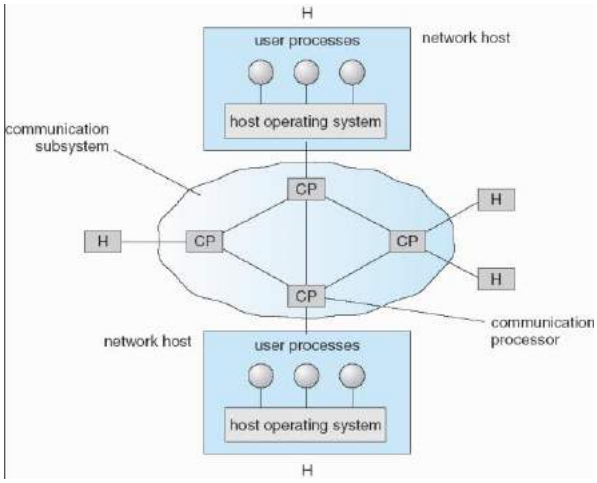


- Wifi is the wireless alternative to Ethernet, but is slower than Ethernet
    - No cables needed to connect hosts to network
- The distance between the wireless router and the host influences the speed of the network

Wide-Area Networks

- WANs emerged mainly as an academic research project
- **Communication processors** control communication links

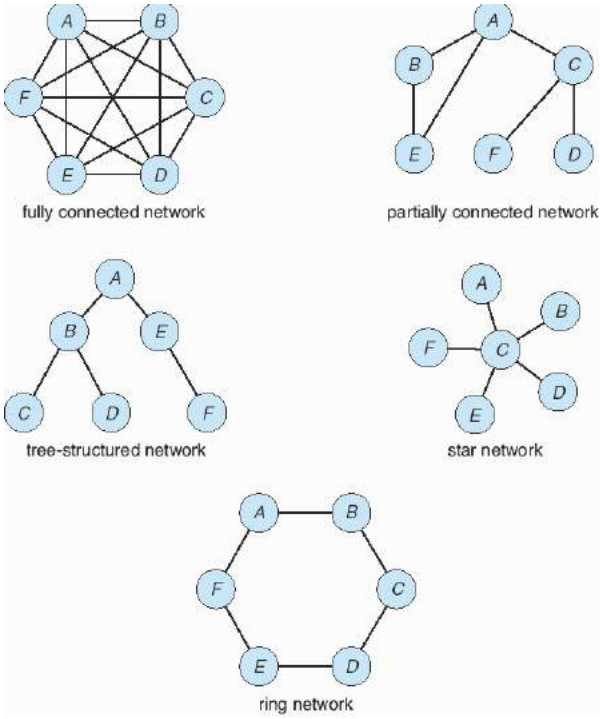
- Responsible for defining the interface through which the sites communicate over the network, as well as for transferring information among the various sites



- Links geographically separated sites
- Point-to-point connections over long-haul lines (often leased from a phone company)
- Speed ≈1.544 to 45Mbps
- Broadcast usually requires multiple messages
- Nodes:
  - usually a high percentage of mainframes

Network Topology

- Sites in the system can be physically connected in a variety of ways; they are compared with respect to the following criteria:
  - **Installation 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*?
  - **Availability/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
- The following six items depict various network topologies:



- The number of links grows as the square of the number of sites, resulting in a huge installation cost
- Criteria for comparing the different configurations:
    - **Installation cost**
        - Low for tree structured networks
        - High for fully connected networks
    - **Communication cost**
        - Low for tree-structured networks
        - Low for star networks
        - High for ring networks
    - **Availability**
        - High for ring networks
        - Lower for tree-structured networks

Communication Structure

- The design of a *communication* network must address five basic *issues:*
    - **Naming and name resolution**
        - How do two processes locate each other to communicate?
    - **Routing strategies**
        - How are messages sent through the network?
    - **Packet strategies**
        - Are packets sent individually or as a sequence?
    - **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

- Each process has an identifier
- Identify processes on remote systems by <host-name, identifier> pair
- The computer's host-name must be **resolved** into a **host-id**
- *Domain name service* (DNS) –specifies the naming structure of the hosts, as well as name-to-address resolution (Internet)
- The OS is responsible for accepting from its process a message destined for **<host-name, identifier>** and for transferring that message to the appropriate host
- The Kernel on the destination host is then responsible for transferring the message to the process named by the identifier
- Check p.686 for DNS protocol

Routing Strategies

- Each site has a routing table, indicating alternative paths
- Fixed routing
  - A path from A to B is specified in advance
  - Disadvantage: Can't adapt to link failures & load changes
- Virtual routing
  - A path from A to B is fixed for the duration of one session
- Dynamic routing
  - The path is chosen only when a message is sent
  - Messages may arrive out of order, so add sequence numbers
- Routers examine the destination Internet address and examine tables to determine the location of the destination host
- With static routing, this table is changed by manual update
- With dynamic routing, a routing protocol is used between routers so that they can update their routing tables automatically

Packet Strategies

- Communication is commonly implemented with fixed-length messages called packets, frames, or datagrams

Connection Strategies

- Once messages are able to reach their destinations, processes can institute **communications sessions** to exchange information
  - Pairs of processes that want to communicate over the network can be connected in a number of ways
  - The three most common schemes are **circuit switching**, **message switching**, and **packet switching**
- **Circuit switching**
  - If two processes want to communicate, a permanent physical link is established between them

        - This link is allocated for the duration of the communication session, and no other process can use that link during this period (even if the two processes are not actively communicating for a while)
        - This scheme is similar to that used in the telephone system
        - Once a communication line has been opened between two parties (that is, party A calls party B), no one else can use this circuit until the communication is terminated explicitly (for example, when the parties hang up)
- **Message switching**
    - If two processes want to communicate, a temporary link is established for the duration of one message transfer
        - Physical links are allocated dynamically among correspondents as needed and are allocated for only short periods
        - Each message is a block of data with system information - such as the source, the destination, and error-correction codes (ECC) - that allows the communication network to deliver the message to the destination correctly
        - This scheme is similar to the post-office mailing system
            - Each letter is a message that contains both the destination address and source (return) address
        - Many messages (from different users) can be shipped over the same link
- **Packet switching**
    - One logical message may have to be divided into a number of packets
        - Each packet may be sent to its destination separately, and each therefore must include a source and a destination address with its data
        - Furthermore, the various packets may take different paths through the network
        - The packets must be reassembled into messages as they arrive
        - Note that it is not harmful for data to be broken into packets, possibly routed separately, and reassembled at the destination
        - Breaking up an audio signal (say, a telephone communication), in contrast, could cause great confusion if it was not done carefully
- There are obvious **tradeoffs** among these schemes:
    - Circuit switching requires substantial setup time and may waste network bandwidth, but it incurs less overhead for shipping each message
    - Conversely, message and packet switching require less set-up time but incur more overhead per message
    - Also, in packet switching, each message must be divided into packets and later reassembled
    - Packet switching is the method most commonly used on data networks because it makes best use of network bandwidth

Contention

- Depending on the network topology, a link may connect more than two sites in the computer network, and several of these sites may want to transmit information over a link simultaneously
    - This situation occurs mainly in a ring or multi-access bus network
    - In this case, the transmitted information may become scrambled
    - If it does, it must be discarded

        - The sites must be notified about there problem so that they can retransmit the information
        - If no special provisions are made, this situation may be repeated, resulting in degraded performance

- Several techniques have been developed to avoid repeated collisions, including **collision detection** and **token passing**
    - **CSMA/CD (carrier sense with multiple access)**:
        - Before transmitting a message over a link, a site must listen to determine whether another message is currently being transmitted over that link
            - If the link is free, the site can start transmitting
            - Otherwise, it must wait (and continue to listen) until the link is free
        - If two or more sites begin transmitting exactly the same time (each thinking that no other site is using the link), then they will register a **collision detection (CD)** and will stop transmitting
            - Each site will try again after some random time interval
        - The main problem with this approach is that, when the system is very busy, many collisions may occur, and thus performance may be degraded
        - Nevertheless, CSMA/CD has been used successfully in the Ethernet system, the most common local area network system
        - One strategy for limiting the number of collisions is to limit the number of hosts per Ethernet network
            - Adding more hosts to a congested network could result in poor network throughput
        - As systems get faster, they are able to send more packets per time segment
            - As a result, the number of systems per Ethernet network generally is decreasing so that networking performance is kept reasonable
    - **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
            - It then removes the token from the ring and begins to transmit its message
            - When the site completes its round of message passing, it retransmits the token
            - This action, in turn, allows another site to receive and remove the token and to start its message transmission
            - If the token gets lost, the system must detect the loss and generate a new token
                - It usually does that by declaring an **election** to choose a unique site where a new token will be generated
        - A token-passing scheme has been adopted by the IBM and HP / Apollo systems
        - The **benefit** of a token-passing network is that performance is constant
        - Adding new sites to a network may lengthen the waiting time for a token, but it will not cause a large performance decrease, as may happen on Ethernet
        - On lightly loaded networks, however, Ethernet is more efficient, because systems can send messages at any time

Communication Protocols

OSI model

- Physical layer
    - Handles mechanical & electrical details of transmission
- Data-link layer
    - Responsible for handling the frames
- Network layer
    - Responsible for providing connections & routing packets
- Transport layer
    - Responsible for low-level access to the network
- Session layer
    - Responsible for implementing sessions
- Presentation layer
    - Responsible for resolving the differences in formats
- Application layer
    - Responsible for interacting directly with the users

Robustness

Failure Detection

- To detect **link** and **site** failure, use a handshaking procedure:
    - At fixed intervals both sites exchange 'I-am-up' messages
    - If site A doesn't receive this message, it can assume
        - That site B has failed, or
        - That the link between A & B has failed, or
        - That the message from B has been lost
    - At this point, site A can
        - Wait for another 'I-am-up' message from B, or
        - Send an 'Are-you-up?' message to B
    - Site A can differentiate between link and site failure by sending an 'Are-you-up?' message by **another** route
    - If B then replies, you know B is up and that the failure is in the direct link between A and B

Reconfiguration

- If a direct **link** from A to B has failed,
    - This info must be broadcast to every site in the system so that the routing tables can be updated accordingly
- If the system believes that a **site** has failed,
    - Every site in the system must be notified, so they will no longer attempt to use the services of the failed site

Recovery from Failure

- If a **link** between A and B has failed,
    - When it is repaired, both A and B must be notified
- If **site** B has failed,
    - When it recovers, it must notify all other sites

Fault Tolerance

Design Issues

An Example: Networking

Summary

Special-Purpose Systems

Real-Time Embedded Systems

- The most prevalent form of computers
- They run embedded real-time OS's that provide limited features

PART EIGHT: SPECIAL PURPOSE SYSTEMS

Chapter 19: Real-Time Systems

Overview

- Rigid time requirements on the operation of a processor
- Sensors bring data to the computer, which must analyze the data and possibly adjust controls to modify the sensor inputs
- E.g. home appliance controllers, weapon & fuel-injection systems
- Processing *must* be done within the time constraints, or it fails
- Hard real-time systems
    - Guarantee that critical tasks are completed on time
    - Data is stored in memory / ROM instead of secondary storage
    - No virtual memory, so no time-sharing
    - No general-purpose OS supports hard real-time functionality
- Soft real-time systems
    - A critical real-time task gets priority over other tasks
    - More limited utility than hard real-time systems
    - Risky to use for industrial control and robotics
    - Useful for multimedia, virtual reality, scientific projects
    - Finding its way into most current OS's, including UNIX

Multimedia Systems

Handheld Systems

- Limited memory, so the OS must manage memory efficiently

- Slow processors, so the OS must not tax the processor
- Small display screens, so web clipping is used for displaying

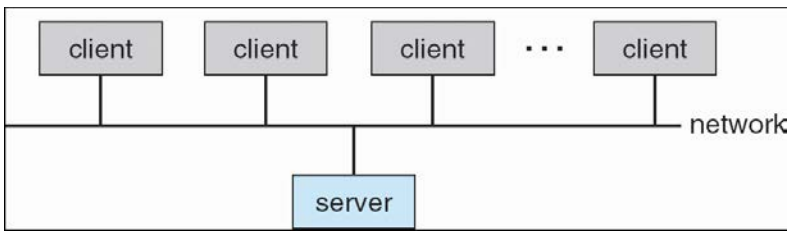
Computing Environments

Traditional Computing

- Companies have portals to provide web access to internal servers
- Network computers are terminals that understand web computing
- Handheld PCs can connect to wireless networks
    - Some homes have firewall to protect them from security breaches
- Traditional computer
    - Blurring over time
    - Office environment
        - PCs connected to a network, terminals attached to mainframe or minicomputers providing batch and timesharing
        - Now portals allowing networked and remote systems access to same resources
    - Home networks
        - Used to be single system, then modems
        - Now firewalled, networked

Client-Server Computing

- Dumb terminals supplanted by smart PCs
- Many systems now servers, responding to requests generated by clients
    - Compute-server provides an interface to client to request services (i.e. database)
    - File-server provides interface for clients to store and retrieve files



Peer-to-Peer Computing

- OS's include system software that enables computers to access the Internet, and several include the web browser itself
- The processors communicate through communication lines
- Network OS = one that provides file sharing across the network
- Another model of distributed system
- P2P does not distinguish clients and servers
    - Instead all nodes are considered peers
    - May each act as client, server or both
    - Node must join P2P network
        - Registers its service with central lookup service on network, or

        - Broadcast request for service and respond to requests for service via *discovery protocol*
    - Examples include *Napster* and *Gnutella*

Web-Based Computing

- PCs are the most prevalent access devices
    - Load balancers distribute network connections
- Web has become ubiquitous
- PCs most prevalent devices
- More devices becoming networked to allow web access
- New category of devices to manage web traffic among similar servers: **load balancers**
- Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients and servers

Chapter 23: Influential Operating Systems

Feature Migration

- Features from mainframes have been adopted by microcomputers

Early Systems

- Early systems were
    - first interactive systems (punch cards!)
    - then became batch systems
    - then interactive again
- Early systems were
    - single-user systems (just one user could work at a time)
    - then became multi-user systems
    - and then single-user systems (with the advent of PCs)
    - and are now aimed at a multi-user environment
    - Dedicated Computer Systems