Note 6

Binding of Instructions and Data to Memory

1. Compile-Time
• Memory location is known before execution.
• Compiler generates absolute code (fixed).
• Program must be recompiled if the starting location changes.

2. Load-Time
• Memory location is not known at compile time.
• Compiler produces relocatable code.

3. Execution-Time
• Address binding happens during program execution.
• Allows the process to move between memory segments while running.
• Requires hardware support.

Logical vs. Physical Address Space

1. Logical Address
• Generated by the CPU.
• Also called a virtual address.

2. Physical Address
• address seen by the memory unit.

3. Relationship Between Them
• Logical and physical addresses are the same in compile-time binding.
• Logical and physical addresses are the same in load-time binding.
• Logical and physical addresses differ in execution-time binding.

Memory-Management Unit (MMU)

1. MMU Function
• Hardware device that maps virtual addresses to physical addresses.

2. Program Perspective
• User programs work only with logical (virtual) addresses.
• Programs never see the actual physical memory addresses.

Swapping

1. Swapping Concept
• A process can be temporarily moved out of memory to a backing store and later brought back to continue execution.

2. Backing Store
• A fast, large disk that holds copies of all users' memory images.
• Must allow direct access to these memory images.

3. Roll Out, Roll In
• A swapping method used in priority scheduling.
• Lower-priority process is swapped out so a higher-priority process can be loaded and executed.

4. Swap Time
• Most of the swap time comes from data transfer.
• Total transfer time increases proportionally with the amount of memory swapped.

Contiguous Allocation

1. Base Register
- Contains the value of the smallest physical address.

2. Limit Register
- Contains the range of logical addresses.

Dynamic Storage-Allocation Problem

1. First-Fit
- Allocate the first hole that is large enough.

2. Best-Fit
- Allocate the smallest hole that is large enough.
- Requires searching the entire list unless it is ordered by size.
- Produces the smallest leftover hole.

3. Worst-Fit
- Allocate the largest hole.
- Also requires searching the entire list.

Fragmentation

1. External Fragmentation
- Total free memory is enough to satisfy a request, but the free space is not contiguous.

2. Internal Fragmentation
- Allocated memory is slightly larger than what was requested.
- The extra space inside the allocated block is unused.

3. Reducing External Fragmentation
- Can be reduced by using compaction.

Paging

1. Noncontiguous Logical Address Space
- A process's logical address space can be noncontiguous.
- The process is placed in any available locations in physical memory.

2. Frames
- Physical memory is divided into fixed-size blocks called frames.

3. Pages
- Logical memory is divided into blocks of the same size called pages.

4. Page Table
- Used to translate logical addresses to physical addresses.

5. Internal Fragmentation
- Occurs within each page.

Address Translation Scheme

1. Page Number (p)
- Used as an index into a page table.

2. Page Offset (d)
- Defines the physical memory address that is sent to the memory unit.

Implementation of Page Table

1. **Page-table base register (PTBR) —>** points to the page table
2. **Page-table length register (PRLR) —>** indicates size of the page table

---

Shared Pages

1. Shared Code
- Must appear in the same location in the logical address space for all processes.

2. Private Code and Data
- Pages for private code and data can appear anywhere in the logical address space.

Structure of the Page Table

1. **Hierarchical Paging**

2. **Hashed Page Tables**

3. **Inverted Page Tables**

Segmentation Architecture

1. Segment Table
- Maps two-dimensional physical addresses.
- Each entry contains:
- **Base –** starting physical address of the segment.
- **Limit –** length of the segment.

2. Segment-Table Base Register (STBR)
- Points to the location of the segment table in memory.

3. Segment-Table Length Register (STLR)
- Indicates the number of segments used by a program.

Note 7

File Concept

Types of Data
- Numeric
- Character
- Binary

File Structure

Simple record structure
- Lines
- Fixed length
- Variable length

Complex structures
- Formatted document
- Relocatable load file

File Attributes

- **Name**: human-readable file name
- **Identifier**: unique number for file identification
- **Type**: file type information
- **Location**: pointer to file location on storage
- **Size**: current file size
- **Protection**: access permissions (read, write, execute)
- **Time/Date/User ID:** used for security, protection, and monitoring

File Operations

- File is an abstract data type (ADT) with a set of operations
- **Create**
- **Write**
- **Read**
- **Reposition within file**
- **Delete**
- **Truncate** (cut off)

Open File Locking

- Mandatory (compulsory) or advisory (consultative)
- **Mandatory**: access is denied based on existing locks and requests
- **Advisory**: processes can check lock status and decide how to proceed

Disk Structure

- Disk is divided into partitions
- Disk or partition can be used raw or formatted with a file system
- A volume is an entity that contains a file system
- Each volume stores file system information in a device directory or volume table of contents

Operations Performed on Directory

1. **Search** for a file
2. **Create** a file
3. **Delete** a file
4. **List** a directory
5. **Rename** a file
6. **Traverse** the file system (tree search)

File System Mounting

• A file system must be mounted before it can be accessed
• An unmounted file system is mounted at a mount point

File Sharing

• File sharing is desirable in multi-user systems
• Sharing is controlled through protection schemes
• In distributed systems, files can be shared over a network

File Sharing – Remote File Systems

• **Client-server** model allows clients to mount remote file systems from servers
• A server can serve multiple clients

Protection

Types of access:

1. **Read**
2. **Write**
3. **Execute**
4. **Append** (attach)
5. **Delete**
6. **List**

Note 8

Ports, busses, device controllers connect to various devices

I/O Hardware

• Signals from I/O devices interface with the computer
• **Port**: connection point for device
• **Bus**: daisy chain or shared direct access
• **Controller** (host adapter): operates port, bus, and device

Direct Memory Access

• Avoids programmed I/O for large data transfers
• Requires a DMA controller
• Transfers data directly between I/O device and memory
• DMA controller takes control of the bus from the CPU

Application I/O Interface

• Each OS has its own I/O subsystem structures and device driver frameworks
• Devices vary in:
• **Character-stream** or **block**
• **Sequential** or **random-access**
• **Synchronous** or **asynchronous**
• **Sharable** or **dedicated**
• **Speed of operation**
• **Read-write, read-only,** or **write-only**

Block and Character Devices

• Block devices include disk drives
• Commands: read, write, seek
• Support raw I/O, direct I/O, or file-system access
• Memory-mapped file access possible
    - Files mapped to virtual memory and loaded via demand paging
• DMA

Clocks and Timers

Programmable interval timer used for timings, periodic interrupts

Blocking and Nonblocking I/O

1. **Blocking**: process suspended until I/O completes
• Easy to use and understand
• Insufficient for some needs

2. **Nonblocking**: I/O returns available data only
• Used for user interface and data copy
• Implemented with multithreading
• Returns immediately with number of bytes read or written
• select() checks readiness, then read() or write() transfers

3. **Asynchronous**: process continues while I/O executes
• Harder to use
• I/O subsystem notifies process when I/O completes

Kernel I/O Subsystem

- **Scheduling**
- **Buffering**: store data in memory while transferring between devices

Kernel I/O Subsystem

- **Caching**: faster device holds copy of data
- Always a copy
- Key to performance
- Sometimes combined with buffering

- **Spooling**: hold output for a device
- Used when device serves one request at a time
- Example: printing

STREAMS

- Full-duplex communication channel between a process and a device
- Used in Unix System V and later

- Each module has:

1. read queue
2. write queue

Improving Performance

- Reduce context switches
- Reduce data copying
- Reduce interrupts (large transfers, smart controllers, polling)
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O for high throughput
- Move user-mode processes or daemons to kernel threads

Note 9

Protection and Security

- **Protection**: mechanisms that control access of processes or users to OS resources
- **Security**: defending the system from internal and external attacks
  - **Attacks include**: denial-of-service, worms, viruses, identity theft, and theft of service

Principles of Protection

- **Guiding principle: principle of least privilege**
- Programs, users, and systems get only needed privileges
- Limits damage from bugs or misuse
- Can be static (system or process lifetime)
- Can be dynamic (changed as needed)
  - Includes domain switching and privilege escalation
- "Need to know" applies to data access
- Rough-grained privilege management is simpler and easier but less precise

Access Matrix

- Protection viewed as a matrix
- Rows = domains
- Columns = objects
- Access(i, j): operations allowed for Domain_i on Object_j

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

Implementation of Access Matrix

- **Option 1: Global table**
- Store triples ⟨domain, object, rights-set⟩
- To check operation M: search for ⟨Di, Oj, Rk⟩ where M ∈ Rk
- Table can be very large
  - May not fit in main memory
- Hard to group objects with common rights

- **Option 2: Access lists for objects**
- Each object has an access list
- List contains pairs ⟨domain, rights-set⟩
  - Shows which domains can access the object
- Can include default access rights

- **Option 3: Capability list for domains**
- Each domain has a list of objects and allowed operations

- **Option 4: Lock-key**
- Compromise between access lists and capability lists
- Each **object** has **locks** (unique bit patterns)
- Each **domain** has **keys** (unique bit patterns)
- Access allowed only if a key matches a lock

Access Control

• Protection applies to non-file resources
• Solaris 10 uses RBAC to enforce least privilege
• Privilege = right to execute a system call or its options
• Privileges assigned to processes
• Users assigned roles with specific privileges and programs
• Role enabled via password to gain privileges
• Similar to access matrix

Comparison of Implementations

1. Global table
- Simple
- Can be very large

2. Access lists
- Match user needs
- Access rights for a domain are hard to determine
- Every object access must be checked
- Many objects and rights → slow

3. Capability lists
- Localize information per process
- Revocation (cancel) can be inefficient

4. Lock-key
- Effective and flexible
- Keys can move between domains
- Easy revocation

• Most systems use a combination of access lists and capabilities

Revocation of Access Rights

1. Access List
• Remove access rights from the access list
• Simple: search list and delete entry
• Can be immediate
• Can be general or selective
• Can be total or partial
• Can be permanent or temporary

2. Capability List
• Requires a scheme to locate the capability
• Capability must be found before it can be revoked

Keys: unique bits associated with capability, generated when capability created
Master key: associated with an object; access allowed when key matches the master key
Revocation: create new master key
Policy: defines who can create and modify keys

Language-Based Protection

- Defining protection policies in a programming language allows a high-level description of how resources are allocated and used.
- The language implementation can enforce protection in software when hardware-based checking is not available.
- Protection specifications are interpreted to generate calls to the protection mechanisms provided by the operating system and hardware.

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);
			...

Stack inspection is a technique for determining the level of trust of currently running code at runtime.

In the example above the untrusted applet's call to get( ) succeeds, because the trusted URL loader asserts the privilege of opening the specific URL lucent.com.

Note 10

The Security Problem

- **Threat**: potential security violation
- **Attack**: attempt to breach security

Security Violation Categories

- **Breach of Confidentiality**: unauthorized reading of data
- **Breach of Integrity**: unauthorized modification of data
- **Breach of Availability**: unauthorized destruction of data
- **Theft of service**: unauthorized use of resources
- **Denial of service (DOS)**: preventing legitimate use

Security Violation Methods

- **Masquerading**: pretending to be an authorized user
- **Replay attack**: reusing (or modifying) captured messages
- **Man-in-the-middle**: attacker intercepts and impersonates both sides
- **Session hijacking**: taking over an active session to bypass authentication

Security Measure Levels

- **Physical**: data centers, servers, terminals
- **Human**: avoid social engineering, phishing, dumpster diving
- **Operating System**: protection mechanisms, debugging
- **Network**: intercepted communications, interruption, DOS

Program Threats

- **Trojan Horse**: code that misuses its environment
- **Trap Door**: hidden access that bypasses normal security
- **Logic Bomb**: Program that initiates a security incident under certain circumstances
- **Stack** and **Buffer Overflow**: exploits program bugs
- **Virus dropper**: inserts virus onto the system
    - Many categories of viruses:
        - File
        - memory
        - Source code
        - Encrypted
        - Polymorphic to avoid having a **virus signature**

Encryption

- A function **E : K → (M→C)** is a function for **generating cipher-texts from messages**
    - E (Encryption)
- A function **D : K → (C → M**) is a function for **generating messages from cipher-texts**
    - D (Decryption)

Passwords

One-time passwords:
- Password computed using a function based on a seed (user and system)
- Generated using hardware device / calculator / key fob
    - Changes very frequently

Implementing Security Defenses

- **Signature-based detection:** spots known bad patterns
- **Anomaly detection:** spots differences from normal behavior

Note 11

Virtual Machines

- **Virtual Machine** : abstract hardware of a single computer into several different execution environments
- Each layer creates a virtual system (virtual machine) for operating systems or applications
- **Host**: the physical hardware system
- **Virtual Machine Manager (VMM)** or **hypervisor**: creates and runs virtual machines with an interface identical to the host
- **Guest**: a virtual copy of the host, usually an operating system

• Implementation of VMMs

- **Type 0 hypervisors**: hardware-based solutions that support VM creation and management via firmware
- **Type 1 hypervisors** (General):operating-system-like software built to provide virtualization
- **Type 1 hypervisors** (Dedicated): also include general-purpose operating systems that provide standard functions as well as VMM functions
- **Type 2 hypervisors**: applications running on a standard operating system but provide VMM features to guest operating systems

• Implementation of VMMs (cont.)

- **Paravirtualization**: guest operating system is modified to cooperate with the VMM to optimize performance
- **Programming-environment virtualization**: creates an optimized virtual system instead of virtualizing real hardware
- **Emulators**: run applications designed for one hardware environment on a different hardware platform
- **Application containment**: segregates applications from the operating system to improve security and management

• OS Component – CPU Scheduling

- CPUs are usually insufficient, leading to CPU overcommitment
- VMM uses standard scheduling algorithms to put threads on CPUs
- Some add fairness aspect

• OS Component – Memory Management

- **Balloon** memory manager communicates with the VMM
- Allocates or deallocates memory to decrease or increase the guest's physical memory usage
- Causes the guest OS to free or have more memory available

• OS Component – I/O

- **Less** hypervisor needs to do for I/O for guests, the **better**

OS Component – Storage Management

- **Physical-to-virtual (P-to-V)** converts native disk blocks into VMM format
- **Virtual-to-physical (V-to-P)** converts virtual format back to native disk format

Note 12

Distributed systems

• **Distributed system**: is a collection of loosely coupled processors interconnected by a communication network
• Processors are referred to as **nodes**, **computers**, **machines**, or **hosts**
• A **site** is the location of a processor
• A **server** provides a resource that a **client node** at a different site wants to use

Reasons for Distributed Systems

• **Resource sharing**
    - Sharing and printing files at remote sites
    - Processing information in a distributed database
    - Using remote specialized hardware devices
• **Computation speedup** – **load sharing** or **job migration**
• **Reliability**
    - detect and recover from site failure, function transfer, reintegrate failed site
• **Communication** – **message passing**
    - All higher-level functions of a standalone system can be expanded to encompass a distributed system
• Computers can be downsized to gain more flexibility, better user interfaces, and easier maintenance using multiple smaller systems

Types of Distributed Operating Systems

1. **Network Operating System**
2. **Distributed Operating Systems**

1. Network-Operating Systems

• Users are aware that multiple machines exist
• Resource access is done explicitly through remote login, Remote Desktop, or File Transfer Protocol (FTP)
• Users must establish sessions and use network-based commands
    - more difficult for users

2. Distributed-Operating Systems

• **Data migration** by transferring entire files or only required file portions
• **Computation migration** by moving computation instead of data
    - Via **remote procedure calls** (RPCs) or **messaging systems**
• **Process migration** allows executing a whole process or parts of it at different sites
    - Load balancing: distributes processes to even the workload
    - Computation speedup: running subprocesses concurrently on different sites
    - Hardware preference: process execution requires a specialized processor
    - Software preference: required software available only at a particular site
    - Data access: running the process remotely instead of transferring all data

Communication Structure

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

Distributed File System

• **Distributed file system (DFS)** is a distributed implementation of a classical time-sharing file system
    - Multiple users share files and storage resources
• DFS manages a set of dispersed storage devices
• Overall storage space is made up of smaller, remotely located storage spaces

DFS Structure

• **Service**: software entity running on one or more machines that provides a particular function to unknown clients
• **Server**: service software running on a single machine
• **Client**: process that invokes a service using a set of operations that forms its client interface

Naming and Transparency

• **Naming** – mapping between logical and physical objects
• **Multilevel mapping** - abstracts a file and hides how and where it is stored on disk
• **Transparent DFS** - hides the network location where the file is stored

Naming Structures

• **Location transparency**: file name does not reveal the physical storage location
• **Location independence**: file name does not change when the file's physical storage location changes

Consistency

• **Client-initiated approach**
    - Client initiates a validity check
    - Server checks consistency with the master copy
• **Server-initiated approach**
    - Server records, for each client, the (parts of) files it caches
    - When server detects a potential inconsistency, it must react