Chapter 1: Introduction

Contents

1 What Operating Systems Do

A computer system can be divided into four parts:

1) **Hardware:** Consists of the CPU, memory and I/O devices. Provides basic computing resources to the system.
2) **Application programs:** Define the ways in which computer resources are to be used to solve user problems.
3) **Operating System (OS):** Software that controls the hardware and coordinates its use among the various application programs for the users.
4) **Users:** The persons or other computers that use this particular computer to solve their problems.

1.1 User View

1) Maximize the work/play that the user is performing.
2) OS designed mainly for **ease of use**. Some attention paid to performance and security and none to **resource utilization** (how various hardware and software resources are shared).
3) Some **embedded computers** in home devices and automobiles have little or no user view. These OSes are designed to run without user intervention.

1.2 System View

1) OS is a **resource allocator**. It addresses the requests and conflicts for resources, and decides how to allocate the resources to specific programs and users so that the computer can be operated efficiently and fairly.

2) A different view: OS is a **control program**. It manages the execution of user programs to prevent errors and improper use of the computer. It is concerned with the operation and control of I/O devices.

1.3 Defining Operating Systems

1) No universally accepted definition. A simple view of an OS is: everything that a vendor ships when you order ”the operating system”. But this varies widly.
2) A more common definition: the OS is the one program running on the computer at all times, usually called the **kernel**.
3) Two other types of programs along with the kernel are:
    a) **System programs:** Associated with the OS but not necessarily part of the kernel.
    b) **Application programs:** Programs not associated with the OS.
4) Mobile OSes contain **middleware**, a set of additional software frameworks that provide services to application developers. Usually support databases, multimedia, graphics, etc.

2 Computer System Organization

1) A modern computer system consists of one or more CPUs and a number of device controllers connected through a common **bus**. It provides access between components and shared memory.
2) Each device controller is in charge of a specific device. A device controller maintains local buffer storage and special-purpose registers.
3) OSes have a **device controller** for each controller. It understands the controller and provides rest of the OS with an interface to the device.
4) Memory controller synchronizes access to memory since device controllers and CPU compete for memory cycles.

2.1 Interrupts

It is the mechanism used by the device controller to inform the device driver that its operation is finished. It is used in modern operating systems to handle asynchronous requests.

2.1.1 Overview:

1) Hardware triggers interrupt by sendingn a signal to the CPU via the system bus.
2) When CPU is interrupted, it stops the current task and transfers execution to a fixed location, which is the starting address of the **interrupt service routine** (ISR).
3) To transfer control to the right ISR, a generic method should be invoked to examine interrupt information. This routine would call the interrupt-specific handler.
4) Interrupts must be handled quickly, as they are frequent. The speed is provided by a table of pointers, called the **interrupt vector**. The starting address of the ISR are indexed by a unique number.
5) The state information prior to the interrupt being triggered is also saved and restored.

2.1.2 Implementation:

1) CPU hardware has an **interrupt-request line** that is sensed after executing every instruction.
2) When a signal is asserted on that line, the CPU reads the interrupt number and uses it as an index into the interrupt vector to jump to the **interrupt-handler routine**.
3) The CPU starts execution from the starting address of the ISR. Necessary states are saved before change and restored prior to returning from the ISR. Execution then continues from the same address before the interrupt was triggered.
4) In summary, the device controller **raises** an interrupt, the CPU **catches** the interrupt and **dispatches** it to the interrupt-handler. The handler **clears** the interrput by servicing the device. This is an **interrupt-driven I/O cycle**.
5) Sophisticated interrupt handling in a modern OS requires, provided by **interrupt-controller hardware**:
    a) Defer interrupt handling during critical signal processing.
    b) Efficient way to dispatch the proper interrupt handler for a device.
    c) Multilevel interrupts to distinguish between the priority of interrupts, and respond with the appropriate urgency.
6) Most CPUs have **nonmaskable** (for unrecoverable memory errors) and **maskable** (can be turned off by the CPU during execution

of critical instructions) interrupt lines. Device controllers use maskable interrupt line.

7) Most CPUs have more interrupts than address elements in the interrupt vector. To accommodate all ISRs, one way is to use **interrupt chaining**.
8) For Intel processors, events 0 to 31 are nonmaskable and events 32 to 255 are maskable. Of the nonmaskable intterupts, 15 and 19 to 31 are **Intel reserved** and should not be used.
9) A system of **interrupt priority levels** enable the CPU to defer handling of low-level priority interrupts without masking all interrupts.

2.2 Storage Structure

1) The CPU can load instructions from memory, hence any programs to be run must be loaded into memory first. Main memory, or **random access memory** (RAM) is implemented using **dynamic random access memory** (DRAM).
2) The first program to run on computer startup is called **bootstrap program**, which loads the OS. However, RAM is **volatile** i.e., it loses its content when power is lost.
3) To store the bootstrap program, the computer uses **electrically erasable programmable read-only memory** (EEPROM) and other forms of **firmware** (nonvolatile storage infrequently written to). These forms of memory are low speed and store static and infrequently used data.
4) A typical instruction executed on a system with a **von Neumann architecture** performs instruction fetch and stores it in the **instruction register**. It is then decoded and the instruction is executed.
5) **Secondary storage** is provided as an extension of main memory. It can hold large quantities of data/programs permanently. Most common examples are **hard-disk drives** (HDD) and **nonvolatile memory devices** (NVM).
6) Many programs use secondary storage as the source and destination of processing. However, it is slower than main memory.
7) **Tertiary storage** includes devices such as magnetic tapes, CD-ROM, etc. that are larger and slower than secondary memory devices. It is used to store backup copies of data and programs.
8) Based on their cost, access time and storage capacity, memory devices are arranged in a **memory hierarchy**.
   a) The volatile memory and NVM are implemented using semiconductor technology.
   b) Nonvolatile storage (NVS) is of two types:
      i) **Mechanical**: HDDs, optical disks, magnetic tape. Chepaer and larger.
      ii) **Electrical**: SSD, NRAM, FRAM. It is also called NVM. Costlier, smaller and faster.

2.3 I/O Structure

1) Interrupt-driven I/O can produce large amount of overhead for bulk data movement. A resolution is **direct memory access** (DMA).
2) In DMA, the device controller sets up buffers, pointers, and counters for the I/O device and then transfers an entire block of data directly to and from the device and main memory. *CPU does not intervene.*
3) Only one interrupt is generated per block rather than per byte. Meanwhile, the CPU performs other work.
4) Some high-end systems use switch rather than bus architecture, where multiple components can operate concurrently. For such systems, DMA is more effective.

3 Computer-System Architecture

Some definitions:

1) **CPU:** The hardware that executes instructions.
2) **Processor:** A physical chip that contains one or more CPUs.
3) **Core:** The basic computation unit of the CPU.
4) **Multicore:** Including multiple computing cores on the same CPU.
5) **Multiprocessor:** Including multiple processors.

3.1 Single-Processor Systems

1) The single-core CPU can run a general-purpose instruction set.
2) These systems have other special-purpose processors running *limited* instruction sets.
3) They are sometiimes managed by the OS, where the OS sends them information about their next task and monitors their status. In

other cases, these processors do their jobs autonomously and consist of low-level components built into hardware.

3.2 Multi-Processor Systems

1) Have at least two general-purpose processors. Increase throughput by a factor slightly smaller than the number of processors (since overheads are incurred in keeping all the parts working correctly and in contention for shared resources).
2) Most common multiprocessor systems use **symmetric multiprocessing systems** (SMP). Each peer CPU performs all tasks.
3) A disadvantage of SMPs is that work may be inefficiently distributed among CPUs. This can be avoided if each CPU shares processes and resources such as memory dynamically.
4) Nowadays, multicore systems are used. Advantages:
   a) On-chip communication faster than between-chip communication.
   b) One multicore chip uses less power compared to multiple single core chips.
5) Each CPU core in a processor has its own local L1 cache and all of the CPU cores in a processor share another L2 cache, which is larger and slower than L1 cache.
6) Adding CPUs to a multiprocessor system does not scale well, since the system bus will become a bottleneck and degrade performance.
7) Instead, CPUs and their local memory are connected via a small, fast local bus called the **shared system interconnect**, so that all CPUs have one physical address space. This is known as the **non-uniform memory approach** (NUMA).
   a) *Advantage*: contention over the system interconnect is avoided when accessing local memory, and NUMA systems scale efficiently when more processors are added.
   b) *Drawback*: increased latency when CPUs must access remote memory across the system interconnect, possible performance penalty.
   c) Increasingly popular on servers/high performance computers.
8) **Blade servers** are systems in which multiple processor boards, I/O boards, networking boards, etc. are placed on the same chassis. Each processor board boots independently and runs its own OS.

3.3 Clustered Systems

1) **Clustered systems** gather multiple CPUs from independently linked **loosely coupled** systems.
2) Used to provide **high-availability service**, that is, services that can be delivered even if one system fails.
3) High availability provides increased reliability.
   a) **Graceful degradation:** Ability to continue provideing service proportional to the level of surviving hardware.
   b) **Fault tolerant:** Systems that can suffer a failure of any single component and still continue operation. Requires a mechanism to detect, diagnose, and correct (if possible) failures.
4) Clusters can be *symmetric* or *asymmetric*.
   a) **Asymmetric clustering:** One machine is in **hot-standby mode** while the other is running the applications. The hot-standby machine monitors the active machine and takes over in case of failure.
   b) **Symmetric clustering:** More than one host is running the application, hosts monitor each other. More efficient, makes most use of available hardware. However, need more than one application to run.
5) Applications designed to run on a cluster are written using a technique known as **parallelization**, to divide a program into separate parts.
6) Some clusters supply access control using a **distributed lock manager** (DLM) to ensure conflicting operations do not occur simultaneously.
7) **Stroage-area networks** (SANs) allow many systems, independent of their location.

4 Operating-System Operations

1) For a computer to start up, the bootstrap program must be run. It initializes CPU registers, device controllers, memory contents, etc. It must know how to locate the OS and load into memory.
2) Once the kernel is loaded, it provides services to the system and its users. Some services

are provided outside the kernel by system programs that are loaded into memory at boot time, these are called **system daemons**.

3) On Linux, the first system program is `systemd` and it starts other daemons.
4) An OS waits for events to happen. Apart from hardware interrupts, there is another form of interrupt called **trap** or **execption**. These are software-generated interrupts that are caused either by error or by a specific request from a user program to perform an OS service by executing a special operation called **system call**.

4.1 Multiprogramming and Multitasking

1) **Multiprogramming** increases CPU utilization and keeps multiple users on one system satisfied.
2) A program in execution in a multiprogrammed system is called a **process**.
3) Some programs may require to wait for the completion of another operation or for user input. In a non-multiprogrammed system, the computer would sit idle, but CPU can switch to another process in multiprogrammed systems.
4) **Multitasking** is an extension of multiprogramming. Here, the CPU frequently switches between multiple processes being executed, providing the user with a fast **response time**.
5) Memory management is needed for several processes in memory. If multiple processes are ready to run at the same time, the system must decide which process to run. This is called **CPU scheduling**.
6) Running multiple processes requires that they should be limited in how they can affect each other, including process scheduling, disk storage and memory management.
7) To ensure reasonable response time, the OS can make use of **virtual memory**, a technique that allows the execution of a process not completely in memory.
   a) Enables users to run programes that are larger than **physical memory**.
   b) Abstracts main memory into a large uniform array of storage, separating **logical memory** as viewed by the user from physical memory.
8) Multiprogramming and multitasking systems must provide a filesystem, storage management, access control to the filesystem, process synchronization, communication, and deadlock management.

4.2 Dual-Mode and Multimode Operation

1) Used to distinguish between execution of operating system code and user-defined code.
2) At least two modes of operation must be supported: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode** or **privileged mode**).
3) To indicate the current mode, a **mode bit** is used. For example, at boot, the hardware starts in kernel mode, the OS is loaded and it starts user applications in user mode.
4) It is a means of protecting OS from errant users and errant users from one another. This is achieved by **privileged instructions**. These are potentially harmful instructions that can be executed only in kernel mode.
5) Executing privileged instructions in user mode will cause the hardware to treat it as illegal and trap to the OS.
6) Can be extended:
   a) Some Intel systems use four separate *protection rings*.
   b) ARMv8 systems have seven modes.
   c) CPUs supporting virtualization have another mode to indicate that the **virtual machine manager** (VMM) is running with privileges between the two modes.
   d) Inital control in a system is with OS. It returns to the OS after executing user applications in user mode. System calls provide the means for a user program to ask the OS to perform privileged operations on behalf of the user.
   e) Execution of system call:
      i) Treated by the hardware as a software interrupt.
      ii) Control passes through interrupt vector to ISR and the mode bit is set to kernel mode.
      iii) Kernel examines instruction that raised the interrupt. Additional information can be provided through registers or in memory via pointers.
      iv) Kernel executes the request after verifying that the arguments are legal, then

passes control to the user.

v) Appropriate error messages are shown and memory dumped for debugging purposes.

4.3 Timer

1) Prevents user program from getting stuck in an infinite loop.
2) May be fixed or variable (implemented by a fixed-rate clock and a counter).
3) Interrupt raised when clock reaches 0.
4) Before giving control to user, OS must ensure that clock is set to interrupt.
5) Instructions that modify timer contents are *privileged*.

5 Resource Management

5.1 Process Management

1) Process required reources such as CPU time, memory, files, and I/O. Allocated while running.
2) Note that program is a **passive** entity and process is an **active** entity.
3) A single-threaded process has only one **program counter** specifying the address of the next instruction to be executed. Multithreaded processes have one program counter for *each* thread.
4) Process: unit of work in a system. System: collection of processes.
5) OS responsible for:
   a) Creating and deleting both user and system processes.
   b) Scheduling processes and threads on CPUs.
   c) Suspending and resuming processes.
   d) Providing mechanisms for process synchronization/communication.

5.2 Memory Management

1) For a program to be executed, it must be mapped to an absolute address in main memory. When it is finished executing, memory must be freed and declared available.
2) Memory management is essential to improve CPU utilization and respose time.
3) OS responsible for:
   a) Keeping track of which parts of memory are being used and which process is using them.
   b) Allocating and deallocating memory space as needed.
   c) Deciding which processes and data to move into and out of memory.

5.3 File-System Management

1) Physical properties of storage devices are abstracted to define a logical storage unit called the **file**. A file is a collection of related information defined by its creator.
2) Files are mapped onto physical media by the OS and accessed via storage devices.
3) Hardware controls the access speed, capacity, data transfer rate and access method.
4) Concept of a file implemented by OS by managing mass storage media and the devices that control them.
5) Files organized into **directories** for ease of use.
6) Read, write and execute access can be implemented by file or directory.
7) OS responsible for:
   a) Creating and deleting files.
   b) Creating and deleting directories to organize files.
   c) Supporting primitives to manipulate files and directories.
   d) Mapping files onto mass storage.
   e) Backup files onto nonvolatile storage media.

5.4 Mass Storage Management

1) Secondary storage (HDD/NVM) is used to back up main memory. Programs are stored on these devices and loaded into memory.
2) Mass storage devices used by these programs as the source and destination of their processing. Proper management of these devices is pivotal.
3) OS responsible for:
   a) Mounting and unmounting
   b) Free-space management
   c) Storage allocation
   d) Disk scheduling
   e) Partitioning
   f) Protection
4) Secondary storage must be used efficiently since it is used frequently and extensively. Speed of operation of a computer is highly dependent on secondary storage and algorithms that manipulate it.

5) Backups can be stored in slower, higher capacity tertiary storage. Not crucial to performance, but must be managed.

5.5 Cache Management

1) **Caching** is the process of copying frequently used data and instructions temporarily to a faster storage system.
2) Internal programmable registers provide a high speed cache for memory. Compilers implement register allocation and replacement algorithms.
3) Other caches such as instruction caches are implemented in main memory. They are not OS controlled.
4) **Cache management** is important since cache size is limited.
5) **Cache coherency** refers to the coherency of the value of a particular memory location stored in various levels of cache.

5.6 I/O System Management

1) The **I/O subsystem** hides peculiarities of the I/O devices. It consists of several components:
   a) Memory-mamgement component that supports buffering, caching, spooling.
   b) General device-driver interface.
   c) Drivers for specific hardware devices.
2) Peculiarities of devices are known only to their respective drivers.

6 Security and Protection

1) **Protection** refers to any mechanism for controlling access of processes or users to the resources defined by the computer system. It must provide means to specify the controls to be imposed and methods to enforce them.
2) Necessity of protection:
   a) Improves reliability by detecting latent errors at the interfaces between component subsystems.
   b) Early detection can prevent contamination of healthy subsystems.
   c) Unprotected resource cannot defend against use by unauthorized or incompetent users.
3) **Security** defends a system from internal and external attacks.
4) Some OSes implement security protocols and others require use of additional software.
5) To distinguish between users on the same system, each user is assigned a unique **user ID** (UID) (called **security ID** (SID) in Windows systems).
6) To distinguish among sets of users, **group identifiers** are used.
7) To gain extra permissions for an activity, a user may have to **escalate privileges**. The process then runs with the **effective UID** until termination.

7 Virutalization

1) **Virtualization** is a technology that allows abstraction of hardware of a single computer into different execution environments.
2) Can be viewed as different OSes running at the same time and can communicate with each other. Each environment consititutes a **virtual machine**.
3) **Emulation** involves simulating computer hardware in software. Typically used when source and target CPU types differ.
4) Examples of emulated systems include *Rosetta*, which allowed applications compiled on IBM CPUs to run on Intel CPUs in Apple systems.
5) Emulation requires translation of machine-level instructions of the source system to the target system. It may run slower than native code.
6) With virtualization, an OS compiled for a particular CPU architecture also runs on another OS for that architecture.
7) The **virutal machine manager** (VMM) manages resources of and protects **guest** OSes from each other.
8) Advantages of virtualization:
   a) Install multiple OSes for exploration.
   b) Install multiple OSes for testing applications for OSes other than the host OS.
   c) Companies use virtualization for running all OSes on a single server for development, testing and debugging.
   d) Datacenters use virtualization to execute and manage multiple computer environments.
9) Examples of VMMs are VMWare ESX, Oracle VirtualBox and Citrix XenServer.

8 Distributed Systems

1) A **distributed system** is a collection of physically separate, possibly heterogeneous, networked computer systems.

2) Users are provided access to the various maintained resources. Access to these shared resources increases computation speed, reliability, functionality and data availability.
3) A **network** is a communication path between two or more systems.
4) Networks vary by protocols used, for example **TCP/IP**.
5) Networks are also characterized by the distances between their nodes:
   a) **Local Area Network** (LAN) connects nodes within a room or a building.
   b) **Wide Area Network** (WAN) links buildings, cities, or countries.
   c) **Metropolitan Area Network** (MAN) could link buildings within a city.
   d) **Personal Area Network** (PAN) links personal devices with very little separation.
6) A **network operating system** is an OS that provides features such as file sharing or communication over a network. Each node acts autonomously, as opposed to distributed OSes.

9 Kernel Data Structures

9.1 Lists, Stacks and Queues

1) A **list** is a collection of data values as a sequence.
2) A **linked list** is a common list implementation. Linked lists can be of various types:
   a) A **singularly linked list**, where each item points to its successor.
   b) A **doubly linked list**, where each item points to its predecessor and successor.
   c) A **circularly linked list**, where each item points to its successor in a circular fashion.
3) A disadvantage of using a linked list is that searching for an element in a linked list of n nodes runs in $O(n)$.
4) Linked lists are used to construct powerful data structures like stacks and queues.
5) A **stack** is a sequentially ordered data structure that uses a last in, first out (LIFO) policy for adding/removing items.
6) The items for insertion and deletion are called **push** and **pop** respectively.
7) Stacks are used when invoking function calls for the address space of each nested function.
8) A **queue** is a sequentially ordered data structure that uses the first in, first out (FIFO) policy for adding/removing items.
9) A queue is used to schedule processes in an OS.

9.2 Trees

1) A **tree** is a data structure that represents data hierarchically.
2) A **general tree** can have nodes with any number of children.
3) A **binary tree** has nodes with at most two children.
4) In a **binary search tree**, an ordering is followed, usually that the left child is smaller than the parent, which is smaller than the right child.
5) However, with binary search trees, the worst-case performance time for searching an element is still $O(n)$.
6) **Balanced binary search trees** are trees in which a tree with n nodes has $\log n$ levels. Thus, searching occurs in $O(\log n)$ time.
7) Linux uses a special kind of tree, called the **red-black** tree for scheduling processes.

9.3 Hash Functions and Maps

1) A **hash function** transforms data to a numeric value after performing operations on it.
2) Finding an element in a hash table can be as quick as $O(1)$.
3) More than one element may have the same hash. To avoid such a **hash collision**, we have a linked list at the table location which contains more than one item.
4) A **hash map** associates key-value pairs using a hash function.
5) Used to hide user credentials.

9.4 Bitmap

1) A **bitmap** is a string of n binary digits that represent the boolean status of each item.
2) Used to save space, or indicate disk block availability.

10 Computing Environments

10.1 Traditional Computing

1) Companies provide **portals** for accessing their servers via the web.

2) **Network computers** or **thin clients** are used in place of traditional workstations to carry around.
3) Web portals can also be accessed by mobile devices using **wireless networks**.
4) A **firewall** prevents networks from security breaches.

10.2 Mobile Computing

1) **Mobile computing** refers to computing on lightweight mobile devices.
2) Used to access the internet, shoot high-quality video, read digital books, etc.
3) Two mobile OSes: Android (Google) and iOS (Apple).

10.3 Client-Server Computing

1) Network architecture can include **server systems** satisfying requests made by **client systems**, thereby creating a **client-server system**.
2) Server systems are of two types:
   a) **Compute-server system** provides an interface for clients to send requests.
   b) **File-serve system** provides a filesystem interface where clients can perform CRUD on files.

10.4 Peer-to-Peer Computing

1) All nodes equal, no distinction.
2) To participate, node must join the network.
3) To find available services:
   a) Register with a centralized lookup service, and contact it when a resource is needed.
   b) Use a **discovery protocol** which broadcasts message to all peers nearby.
4) Examples include Napster, Skype (hybrid peer-to-peer) using **voice over IP** (VoIP).

10.5 Cloud Computing

1) **Cloud computing** delivers computing, storage and apps as a service across a network. It is a logical extension of virtualization.
2) Types of cloud computing:
   a) **Public cloud**, available an pay via Internet.
   b) **Private cloud**, run by a company for personal use.
   c) **Hybrid cloud**, inclues components from the other two types.
   d) Software as a Service (SaaS), where softwares are available over the Internet.
   e) Platform as a Service (PaaS), which gives a software stack ready for application use via the Internet.
   f) Infrastructure as a Service (IaaS), where servers or storage is available over the Internet.

10.6 Real-Time Embedded Systems

1) Embedded computers most prevalent, found in many appliances.
2) Considerable variation, depending on functionality. Some have application specific integrated circuits (ASICs) to perform their tasks without OS.
3) Becoming useful with Web access and Internet of Things (IoT).
4) Embedded systems run **real time operating systems**, where rigid time requirements are placed on the processor or flow of data. For instance, medical systems, automobile engine/braking systems.
5) Well-defined, fixed time constraints for the system to satisfy, to prevent failure.