What is an operating system?

While there is no universally accepted definition, an operating system is a program that acts as intermediary between user and hardware. Major goals:

- Execute programs and make problem solving easier
- Make computer system convenient to use
- Use hardware in efficient manner

It is the one program running at all times, usually called a **kernel**.

The main difference between mainframe computer and personal computers operating systems are that PCs allow for wasted resources (such as CPU cycles) to improve usability and functionality. Mainframes maximize resource use at the expense of usability.

Computer System Structure

Can be divided into four components:

- Hardware: (CPU, memory, I/O) provides basic computing resources
- Operating System: Controls/coordinates hardware among applications/users
- Application Programs: define ways in which resources used to solve problems
- Users: people, machines, or other computers

Computer Startup

The initial program that runs at power-up or reboot is called a **bootstrap** program, or firmware. It is stored in the ROM and initializes all aspects of the system. It is responsible for loading and executing the kernel.

Interrupts

An operating system is **interrupt** driven. Interrupts are used to signal the occurrence of an event, either from the hardware (by sending signal to the CPU) or software (by executing a system call, traps, or exceptions). An interrupt transfers control to the interrupt service routine through an interrupt vector that contains the addresses of the service routines. It will save the address of the interrupted instruction and disable incoming interrupts while being processed.

Storage Structure

- Main memory: large storage media CPU can access directly (RAM). Volatile.
- Secondary storage: nonvolatile extension of main memory.
- Hard Disks: magnetic non-volatile storage medium (ROM)

Storage is arranged in a hierarchy, with the fastest, costliest, and most volatile storage mediums at the top ranging from registers to the hard disk to optical/magnetic storage.

Caching is the copying of information from slower to faster storage. Main memory can be thought of as a cache for secondary storage. First the faster storage is checked to see if information is there, if so it is used if not it is copied.

Computer Architecture

Many systems use a single general-purpose processor, but **multiprocessor** systems are growing in use and importance. The advantages include:

- Increased throughput
- Economy of scale
- Increased reliability (fault tolerance)

A disadvantage is that they are more complex and require additional hardware.

The two types of multiprocessor systems are **asymmetric** (where each processor is assigned a specific task, master-slave) and **symmetric** (each processor performs all tasks, peer network)

Clustered Systems

Like multiprocessor systems but with multiple systems working together. Usually share storage via **storage-area network (SAN)**. Provides high availability service which survives failures. Asymmetric clustering has one machine in hot-standby mode, while symmetric clustering has multiple nodes running applications monitoring each other. Applications must be written using parallelization.

Operating System Structure

Multiprogramming is needed for efficiency, so that jobs are organized so that CPU always has one to execute. The subset of total jobs is kept in memory and selected to run via job scheduling. When that job has to wait the OS switches to another job.

Timesharing/multitasking is an extension of multiprogramming. This allows many people at various terminals to use a computer system at the same time. Possible security problems include stealing or copying program or data, or using resources without proper allocation. Cannot ensure same degree of security as in a dedicated machine.

Operating System Operations

As mentioned an operating system is interrupt driven, with hardware interrupts from devices and software interrupts from exceptions or traps. Problems such as infinite loops can be handled by the use of timer interrupts. The problem of processes modifying each other in the OS can be handled by dual mode, which is where the OS uses a user mode and a kernel mode. Instructions are **privileged** if they are only executable in kernel mode. This protects the system from errant users, and users from one another.

Major OS Concepts/Components/Functions

Process Management

A **process** is a program in execution. Compared to a program, which is a passive entity, a process is an active entity which is a unit of work within the system. A process executes instructions sequentially until completion, using resources to accomplish its task. Typically many processes are running concurrently, and each releases resources upon termination.

The five major activities of an OS in regard to process management are:

- Create and delete processes
- Suspend/resume processes
- Mechanism for process synchronization
- Mechanism for process communication
- Mechanism for deadlock handling

Memory Management

Main memory is a large array of words or bytes, each with its own address. This is storage that can be accessed directly by the CPU. For a program to be executed it must be in the main memory.

Memory management activities include: keeping track of what parts of the memory are being used and by whom, deciding which processes are loaded into memory when space is available, and allocating/deallocating space as needed.

File Management (Storage disk management)

Files are collections of related information defined by its creator. They are stored on the disk, allowing long term storage. The five major activities of an OS in regard to file management are:

- Creation and deletion of files
- Creation and deletion of directories
- Support of primitives for manipulating files and directories
- Mapping of files onto secondary storage
- Backup of files on stable storage media

Mass-Storagement management allows for the managing of free space on secondary storage devices, allocation of storage space to write new files, and scheduling requests for memory access.

I/O Control and Management

Hides the peculiarities of hardware devices from the users. Responsible for: memory management of I/O (including buffering, caching, spooling), general device driver interface, and drivers for specific hardware devices.

Protection and Security

Protecting is any mechanism for controlling access of programs, processes, or users to resources defined by OS. Security is the defense of the system against internal/external attacks.

Computing Environments

Traditional: stand-alone general purpose machines.
Client-Server: Dumb terminals supplanted by smart PCs. Database interface.
Distributed Computing: Networked systems that distribute computing and resources.
Peer-to-Peer: node connected peers that can each act as client, server, or both.
Web-based: applications that run over web servers and clients.
Virtualization: operating systems running inside other OSes.
Cloud: internet based computing, load balancers spread traffic across applications
Open-source OS: Linux, Unix, etc.

Types of networks: LAN, MAN, WAN

Operating System Structures

Operating System Services two sets: those that are helpful to user, and those that ensure efficient operation of the system.

Helpful to user:

- **User Interface:** GUI or command line to help navigate
    - **Command Line:** allows direct command entry that reads commands from the user or file of commands and executes them. Usually not part of the kernel. Allows user to create and manage processes and also determine ways by which they communicate (such as through pipes and files)
    - **GUI:** implemented as a desktop metaphor (file folders, trash cans, etc)
    - **Touchscreen:** when mouse not possible. Actions based on gestures
- **Program Execution:** allows user to execute programs without worrying about memory allocation or multitasking
- **I/O Operations:** hides details of underlying hardware
- **File System Manipulation:** read and write files and directories, create and delete them, search them, list information, handle permission management
- **Communication:** process information exchange on same computer or network
- **Error Detection:** aware of possible errors to take action to ensure correct and consistent computing

Ensure efficient system operation:

- **Resource Allocation:** allocates CPU cycles, memory, file storace, and I/O when multiple users or multiple jobs are running concurrently
- **Accounting:** tracks how much/what kind of resources each user is using
- **Protection/Security:** access control to resources, authentication, and other mechanisms to control the use of information on a multiuser or networked system

System Calls and System Programs

System calls are the programming interface to the services provided by the OS. Typically written in a high-level language (C or C++) and mostly accessed by programs via high-level Application Program Interface (API) rather than direct system call use.

There is typically a number associated with each system call and the interface maintains a table indexed according to those numbers. The interface invokes the system call in OS kernel and returns the status of the system call and any return values. The caller doesn't need to know how the calls are implemented.

For parameter passing there are three general methods used to pass parameters to OS:

- Simplest: pass in registers
- Store in a block, or table, in memory and pass address of block
- Parameters pushed onto stack by program and popped off by OS

The block and stack methods do not limit the number or length of parameter passed.

Types of system calls can be grouped into six major categories:

- **Process Control:** load, execute, end, abort, create process, get/set process attributes, wait for time/signal, allocate/free memory
- **File Management:** Request/release device, read/write data, get/set file attributes
- **Device Management:** Request/release device, read/write data, get/set attributes
- **Information Maintenance:** get/set time or date, get/set system data, get/set attributes for process/file/device
- **Communications:** create/delete connection, send/receive messages, attach/detach devices
- **Protection:** control access to resources, get/set permissions, allow/deny user access

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System programs provide convenient environment for development and execution:

- **File Management** (create, delete, copy, rename, print, list, etc)
- **Status Information** (date, time, memory, disk space, users, etc)
- **File Modification** (text editors, file search)
- **Programming Language Support** (compiler, linkers, interpreters)
- **Program Loading and Execution** (loaders)
- **Communications** (virtual links, send/receive messages)
- **Application Programs** (web browsers, office suites)

Most user's view of the operating system is defined by system programs, not the actual system calls.

OS Design and Implementation

Internal structure of different OS can vary widely. Design should have user goals (convenient to use, easy to learn, reliable, safe, fast) and system goals (easy to design, implement, and maintain. Flexible, reliable, error-free, efficient). Various structures:

- **Simple:** written to provide most functionality in least space (MS-DOS)
    - Not divided into modules
    - Single tasking systems
    - Shell invoked when system booted
    - Simple method to run program (no process created)
    - Single memory space
    - loads program into memory overwriting all but kernel
    - program exit reloads shell
- **Not Simple:** Example is UNIX. OS consists of two separable parts: systems programs and the kernel.
- **Layered:** OS is divided into a number of levels. Bottom is the hardware with top being the user interface. Modularity allows layers to be selected such that each uses functions and services of only lower level layers.
- **Microkernel:** moves a lot of the kernel into user space. Communication takes place between user modules using message passing. The benefits are: it is easier to extend a microkernel, easier to port the OS to new architectures, more reliable, more secure. Disadvantage is that performance overhead of user space to kernel space communication.

Most modern OS are not one pure model, but are hybrid.

Many modern OS implement **loadable kernel modules**. These: use object-oriented approach, separates each core component, uses known interfaces to talk to the others, and is loadable as needed within the kernel. It is similar to a layered system in that each kernel section has defined, protected interfaces, but is more flexible than a layered system because any module can call any other module.

Virtual machines treat hardware and the OS kernel as though they were all hardware. The OS creates the illusion of multiple processes, with each executing on its own processor with its own (virtual) memory. They are different OS that can share the same hardware. Some file sharing can be permitted and controlled. They can communicate with each other and other physical systems via network. This makes them useful for development, testing, and research as they are easy to debug and security problems are easy to solve.

This concept provides complete protection of system resources since each VM is isolated from other VMs. This prevents direct sharing of resources (indirect through a network, internet, etc)