

OS Part-1

- crafted by Love Babbar and Lakshay Kumar



Application software performs specific task for the user.

System software operates and controls the computer system and provides a platform to run application software.

An **operating system** is a piece of software that manages all the resources of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner by hiding underlying complexity of the hardware and acting as a resource manager.

Why OS?

1. What if there is no OS?
 - a. Bulky and complex app. (Hardware interaction code must be in app's code base)
 - b. Resource exploitation by 1 App.
 - c. No memory protection.
2. What is an OS made up of?
 - a. Collection of system software.

An operating system function -

- Access to the computer hardware.
- interface between the user and the computer hardware
- **Resource management (Aka, Arbitration) (memory, device, file, security, process etc)**
- **Hides the underlying complexity of the hardware. (Aka, Abstraction)**
- facilitates execution of application programs by providing isolation and protection.

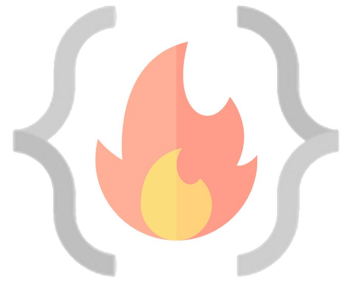


User

Application programs
Operating system
Computer hardware

The operating system provides the means for proper use of the resources in the operation of the computer system.

Types of OS

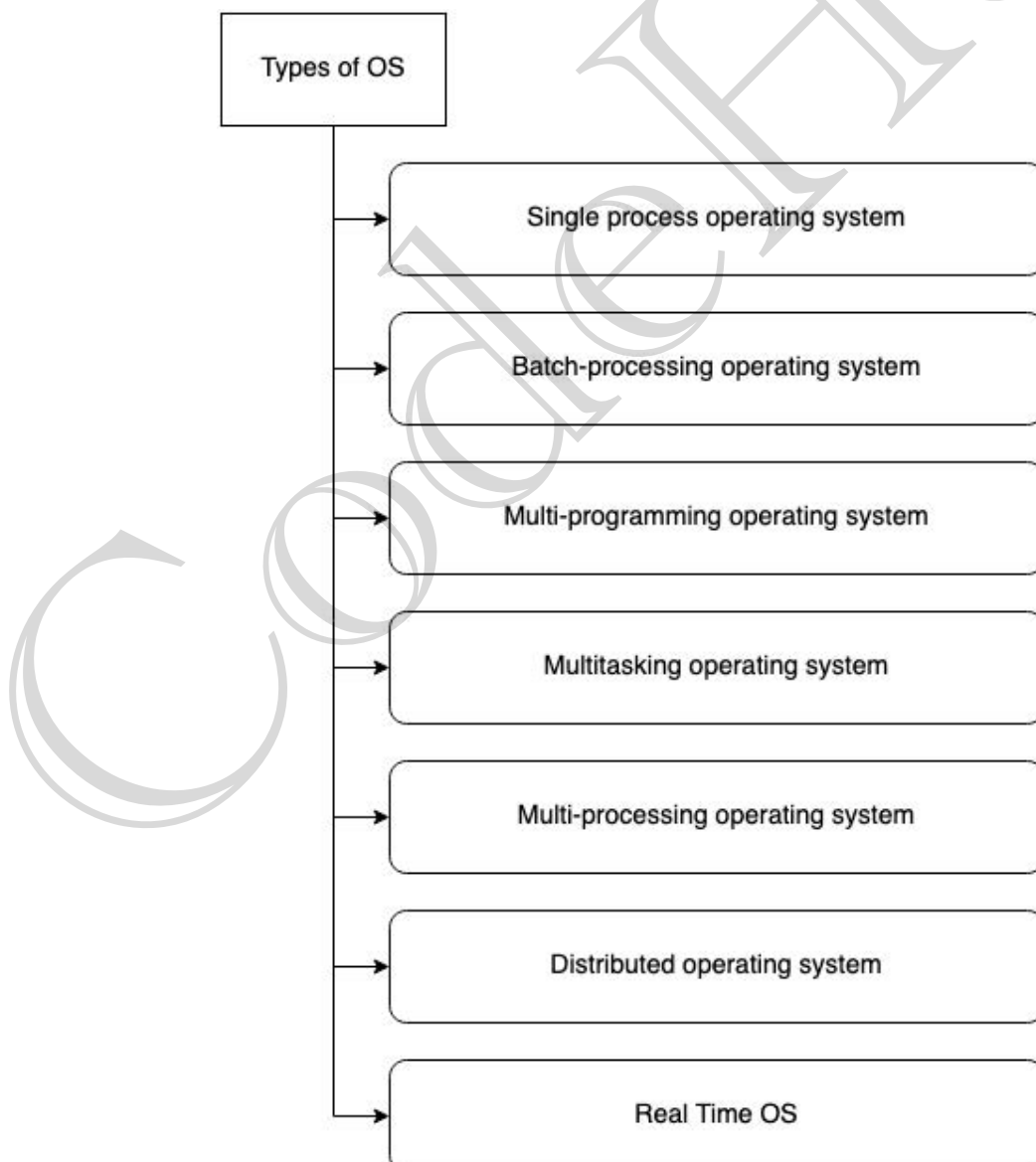


OS goals –

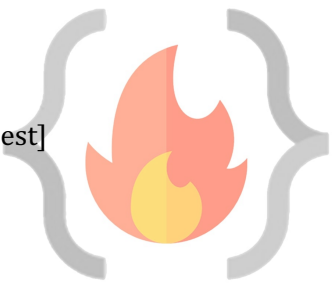
- Maximum CPU utilization
- Less process starvation
- Higher priority job execution

Types of operating systems –

- | | |
|-------------------------------------|---|
| - Single process operating system | [MS DOS, 1981] |
| - Batch-processing operating system | [ATLAS, Manchester Univ., late 1950s – early 1960s] |
| - Multiprogramming operating system | [THE, Dijkstra, early 1960s] |
| - Multitasking operating system | [CTSS, MIT, early 1960s] |
| - Multi-processing operating system | [Windows NT] |
| - Distributed system | [LOCUS] |
| - Real time OS | [ATCS] |

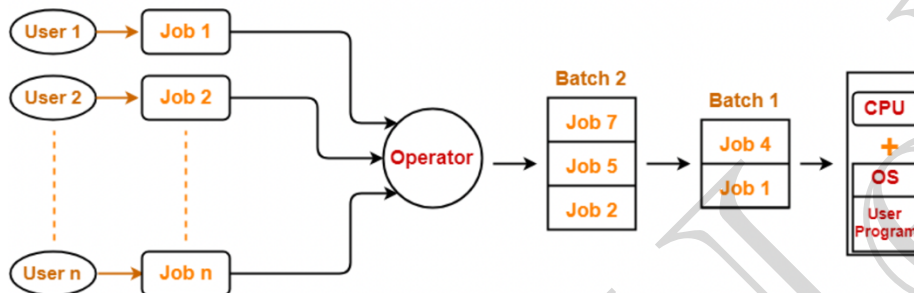


Single process OS, only 1 process executes at a time from the ready queue. [Oldest]



Batch-processing OS,

1. Firstly, user prepares his job using punch cards.
 2. Then, he submits the job to the computer operator.
 3. Operator collects the jobs from different users and sort the jobs into batches with similar needs.
 4. Then, operator submits the batches to the processor one by one.
 5. All the jobs of one batch are executed together.
- Priorities cannot be set, if a job comes with some higher priority.
 - May lead to starvation. (A batch may take more time to complete)
 - CPU may become idle in case of I/O operations.



Multiprogramming increases CPU utilization by keeping multiple jobs (code and data) in the **memory** so that the CPU always has one to execute in case some job gets busy with I/O.

- Single CPU
- Context switching for processes.
- Switch happens when current process goes to wait state.
- CPU idle time reduced.

Multitasking is a logical extension of multiprogramming.

- Single CPU
- Able to run more than one task simultaneously.
- Context switching and time sharing used.
- Increases responsiveness.
- CPU idle time is further reduced.

Multi-processing OS, more than 1 CPU in a single computer.

- Increases reliability, 1 CPU fails, other can work
- Better throughput.
- Lesser process starvation, (if 1 CPU is working on some process, other can be executed on other CPU).



Distributed OS,

- OS manages many bunches of resources, ≥ 1 CPUs, ≥ 1 memory, ≥ 1 GPUs, etc
- **Loosely connected autonomous**, interconnected computer nodes.
- collection of independent, networked, communicating, and physically separate computational nodes.

RTOS

- **Real time** error free, computations within tight-time boundaries.
- Air Traffic control system, ROBOTS etc.

CodeHelp



Multi-Tasking vs Multi-Threading

Program: A Program is an executable file which contains a certain set of instructions written to complete the specific job or operation on your computer.

- It's a compiled code. Ready to be executed.
- Stored in Disk

Process: Program under execution. Resides in Computer's primary memory (RAM).

Thread:

- Single sequence stream within a process.
- An independent path of execution in a process.
- Light-weight process.
- Used to achieve parallelism by dividing a process's tasks which are independent path of execution.
- E.g., Multiple tabs in a browser, text editor (When you are typing in an editor, spell-checking, formatting of text and saving the text are done concurrently by multiple threads.)

Multi-Tasking	Multi-Threading
The execution of more than one task simultaneously is called as multitasking.	A process is divided into several different sub-tasks called as threads, which has its own path of execution. This concept is called as multithreading.
Concept of more than 1 processes being context switched.	Concept of more than 1 thread. Threads are context switched.
No. of CPU 1.	No. of CPU ≥ 1 . (Better to have more than 1)
Isolation and memory protection exists. OS must allocate separate memory and resources to each program that CPU is executing.	No isolation and memory protection , resources are shared among threads of that process. OS allocates memory to a process; multiple threads of that process share the same memory and resources allocated to the process.

Thread Scheduling:

Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.

Difference between Thread Context Switching and Process Context Switching:

Thread Context switching	Process context switching
OS saves current state of thread & switches to another thread of same process.	OS saves current state of process & switches to another process by restoring its state.



Doesn't includes switching of memory address space. (But Program counter, registers & stack are included.)	Includes switching of memory address space.
Fast switching.	Slow switching.
CPU's cache state is preserved.	CPU's cache state is flushed.

CodeHelp

Components of OS



1. **Kernel:** A **kernel** is that part of the operating system which interacts directly with the hardware and performs the most crucial tasks.
 - a. Heart of OS/Core component
 - b. Very first part of OS to load on start-up.
2. **User space:** Where application software runs, apps don't have privileged access to the underlying hardware. It interacts with kernel.
 - a. GUI
 - b. CLI

A **shell**, also known as a command interpreter, is that part of the operating system that receives commands from the users and gets them executed.

Functions of Kernel:

1. **Process management:**
 - a. Scheduling processes and threads on the CPUs.
 - b. Creating & deleting both user and system process.
 - c. Suspending and resuming processes
 - d. Providing mechanisms for process synchronization or process communication.
2. **Memory management:**
 - a. Allocating and deallocating memory space as per need.
 - b. Keeping track of which part of memory are currently being used and by which process.
3. **File management:**
 - a. Creating and deleting files.
 - b. Creating and deleting directories to organize files.
 - c. Mapping files into secondary storage.
 - d. Backup support onto a stable storage media.
4. **I/O management:** to manage and control I/O operations and I/O devices
 - a. Buffering (data copy between two devices), caching and spooling.
 - i. Spooling
 1. Within differing speed two jobs.
 2. Eg. Print spooling and mail spooling.
 - ii. Buffering
 1. Within one job.
 2. Eg. Youtube video buffering
 - iii. Caching
 1. Memory caching, Web caching etc.

Types of Kernels:

1. Monolithic kernel
 - a. All functions are in kernel itself.
 - b. **Bulky in size.**
 - c. **Memory required to run is high.**
 - d. **Less reliable, one module crashes -> whole kernel is down.**
 - e. High performance as communication is fast. (Less user mode, kernel mode overheads)
 - f. Eg. Linux, Unix, MS-DOS.

2. Micro Kernel

- a. Only major functions are in kernel.
 - i. Memory mgmt.
 - ii. Process mgmt.
- b. File mgmt. and IO mgmt. are in User-space.
- c. smaller in size.
- d. More Reliable
- e. More stable
- f. Performance is slow.
- g. Overhead switching b/w user mode and kernel mode.
- h. Eg. L4 Linux, Symbian OS, MINIX etc.



3. Hybrid Kernel:

- a. Advantages of both worlds. (File mgmt. in User space and rest in Kernel space.)
- b. Combined approach.
- c. Speed and design of mono.
- d. Modularity and stability of micro.
- e. Eg. MacOS, Windows NT/7/10
- f. IPC also happens but lesser overheads

4. Nano/Exo kernels...

Q. How will communication happen between user mode and kernel mode?

Ans. Inter process communication (**IPC**).

1. Two processes executing independently, having independent memory space (Memory protection), But some may need to communicate to work.
2. Done by shared memory and message passing.

System Calls



How do apps interact with Kernel? -> using system calls. Eg. Mkdir laks

- Mkdir indirectly calls kernel and asked the file mgmt. module to create a new directory.
- Mkdir is just a wrapper of actual system calls.
- Mkdir interacts with kernel using system calls.

Eg. Creating a process.

- User executes a process. (User space)
- Gets system call. (US)
- Exec system call to create a process. (KS)
- Return to US.

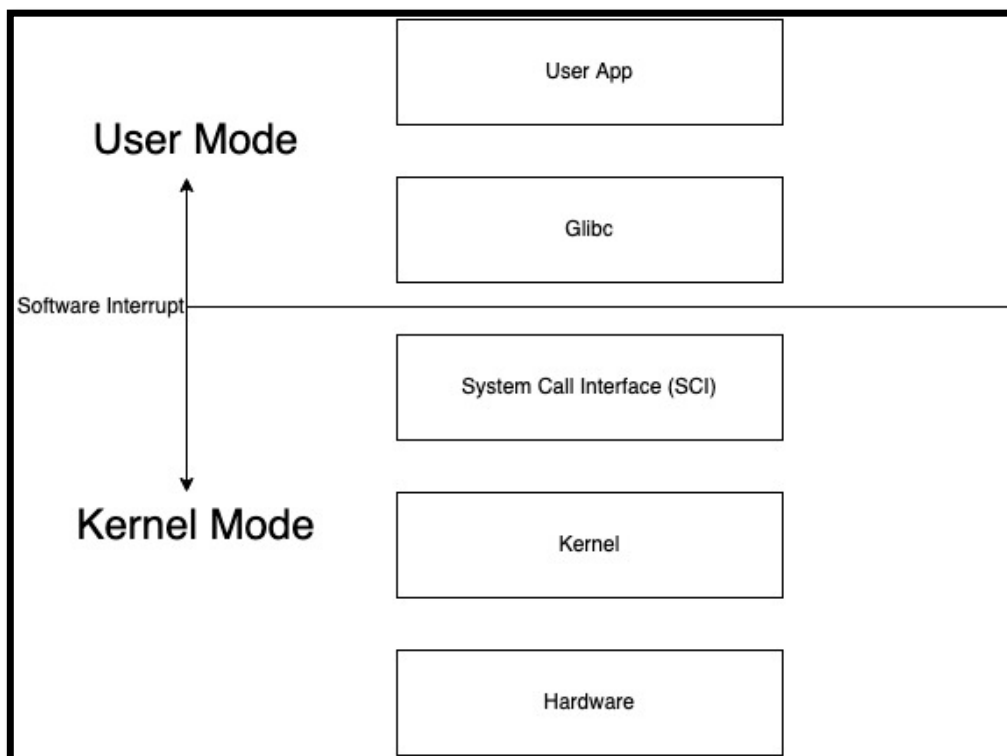
Transitions from US to KS done by software interrupts.

System calls are implemented in C.

A **system call** is a mechanism using which a user program can request a service from the kernel for which it does not have the permission to perform.

User programs typically do not have permission to perform operations like accessing I/O devices and communicating other programs.

System Calls are the only way through which a process can go into **kernel mode from user mode**.





Types of System Calls:

- 1) Process Control
 - a. end, abort
 - b. load, execute
 - c. create process, terminate process
 - d. get process attributes, set process attributes
 - e. wait for time
 - f. wait event, signal event
 - g. allocate and free memory
- 2) File Management
 - a. create file, delete file
 - b. open, close
 - c. read, write, reposition
 - d. get file attributes, set file attributes
- 3) Device Management
 - a. request device, release device
 - b. read, write, reposition
 - c. get device attributes, set device attributes
 - d. logically attach or detach devices
- 4) Information maintenance
 - a. get time or date, set time or date
 - b. get system data, set system data
 - c. get process, file, or device attributes
 - d. set process, file, or device attributes
- 5) Communication Management
 - a. create, delete communication connection
 - b. send, receive messages
 - c. transfer status information
 - d. attach or detach remote devices

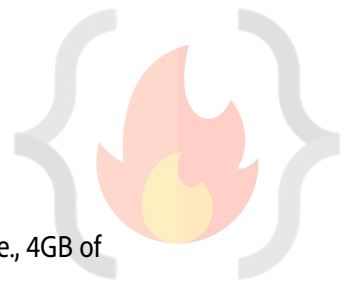
Examples of Windows & Unix System calls:

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

What happens when you turn on your computer?



- i. PC On
- ii. CPU initializes itself and looks for a firmware program (BIOS) stored in BIOS Chip (Basic input-output system chip is a ROM chip found on mother board that allows to access & setup computer system at most basic level.)
 1. In modern PCs, CPU loads UEFI (Unified extensible firmware interface)
- iii. **CPU** runs the BIOS which tests and initializes system hardware. Bios loads configuration settings. If something is not appropriate (like missing RAM) error is thrown and boot process is stopped.
This is called **POST** (Power on self-test) process.
(UEFI can do a lot more than just initialize hardware; it's really a tiny operating system. For example, Intel CPUs have the [Intel Management Engine](#). This provides a variety of features, including powering Intel's Active Management Technology, which allows for remote management of business PCs.)
- iv. **BIOS** will handoff responsibility for booting your PC to your OS's bootloader.
 1. BIOS looked at the [MBR \(master boot record\)](#), a special boot sector at the beginning of a disk. The MBR contains code that loads the rest of the operating system, known as a "bootloader." The BIOS executes the bootloader, which takes it from there and begins booting the actual operating system—Windows or Linux, for example.
In other words,
the BIOS or UEFI examines a storage device on your system to look for a small program, either in the MBR or on an EFI system partition, and runs it.
- v. The bootloader is a small program that has the large task of booting the rest of the operating system (Boots Kernel then, User Space). Windows uses a bootloader named Windows Boot Manager (Bootmgr.exe), most Linux systems use [GRUB](#), and Macs use something called boot.efi



32-Bit vs 64-Bit OS

1. A 32-bit OS has 32-bit registers, and it can access 2^{32} unique memory addresses. i.e., 4GB of physical memory.
2. A 64-bit OS has 64-bit registers, and it can access 2^{64} unique memory addresses. i.e., 17,179,869,184 GB of physical memory.
3. 32-bit CPU architecture can process 32 bits of data & information.
4. 64-bit CPU architecture can process 64 bits of data & information.
5. Advantages of 64-bit over the 32-bit operating system:
 - a. **Addressable Memory:** 32-bit CPU $\rightarrow 2^{32}$ memory addresses, 64-bit CPU $\rightarrow 2^{64}$ memory addresses.
 - b. **Resource usage:** Installing more RAM on a system with a 32-bit OS doesn't impact performance. However, upgrade that system with excess RAM to the 64-bit version of Windows, and you'll notice a difference.
 - c. **Performance:** All calculations take place in the registers. When you're performing math in your code, operands are loaded from memory into registers. So, having larger registers allow you to perform larger calculations at the same time.
32-bit processor can execute 4 bytes of data in 1 instruction cycle while 64-bit means that processor can execute 8 bytes of data in 1 instruction cycle.
(In 1 sec, there could be thousands to billions of instruction cycles depending upon a processor design)
 - d. **Compatibility:** 64-bit CPU can run both 32-bit and 64-bit OS. While 32-bit CPU can only run 32-bit OS.
 - e. **Better Graphics performance:** 8-bytes graphics calculations make graphics-intensive apps run faster.