Great Place To Work. Certified FEB 2023 – FEB 2024 INDIA

UPES

UNIVERSITY OF TOMORROW

nirf RANKING 2023

#RANKED 52 IN INDIA

#University Category

QS WORLD UNIVERSITY RANKINGS

NO.1 PVT. UNIVERSITY IN ACADEMIC REPUTATION IN INDIA

ACCREDITED WITH GRADE A NAAC

ACCREDITED **GRADE 'A'** BY NAAC

E-LEAD

PERFECT SCORE OF **150/150** AS A TESTAMENT TO EXECEPTIONAL E-LEARNING METHODS

Unit 1 : Introduction to Operating System

Lecture 6

Dr. Hemant Petwal

School of Computer Science

UPES, Dehradun

India

UPES
UNIVERSITY OF TOMORROW

Table of Contents

UPES
UNIVERSITY OF TOMORROW

Learning & Course Outcomes

Learning Outcomes

LO1: Understand the concept of System call in Operating System

LO2: Understand booting process, different types of system calls and open-source operating systems

Course Outcomes

CO1: Demonstrate a comprehensive understanding of operating systems

UPES
UNIVERSITY OF TOMORROW

System Booting

Booting is the process of starting a computer. It can be initiated by hardware such as a button press or by a software command.

After it is switched on, a **CPU has no software in its main memory, so some processes must load software into memory before execution**. This may be done by hardware or firmware in the CPU or by a separate processor in the computer system.

Booting Operation Sequence:

- Booting is a start-up sequence that starts the operating system of a computer when it is turned on.
- A boot sequence is the initial set of operations that the computer performs when it is switched on. Every computer has a boot sequence.

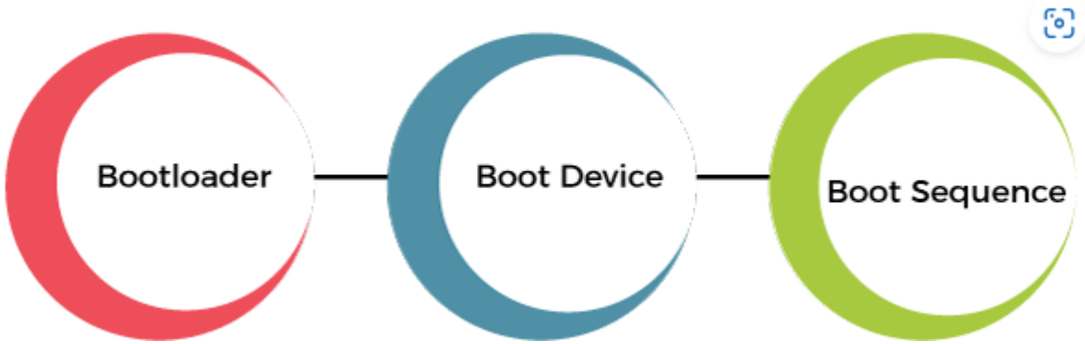


Fig 1. Booting Operation Sequence

UPES
UNIVERSITY OF TOMORROW

System Booting

When a computer or any other computing device is in a powerless state, its operating system remains stored in secondary storage like a hard disk or SSD. But, when the computer is started, the operating system must be present in the main memory or RAM of the system.

When a computer system is started, there is a mechanism in the system that loads the operating system from the secondary storage into the main memory, or RAM, of the system. This is called the booting process of the system.

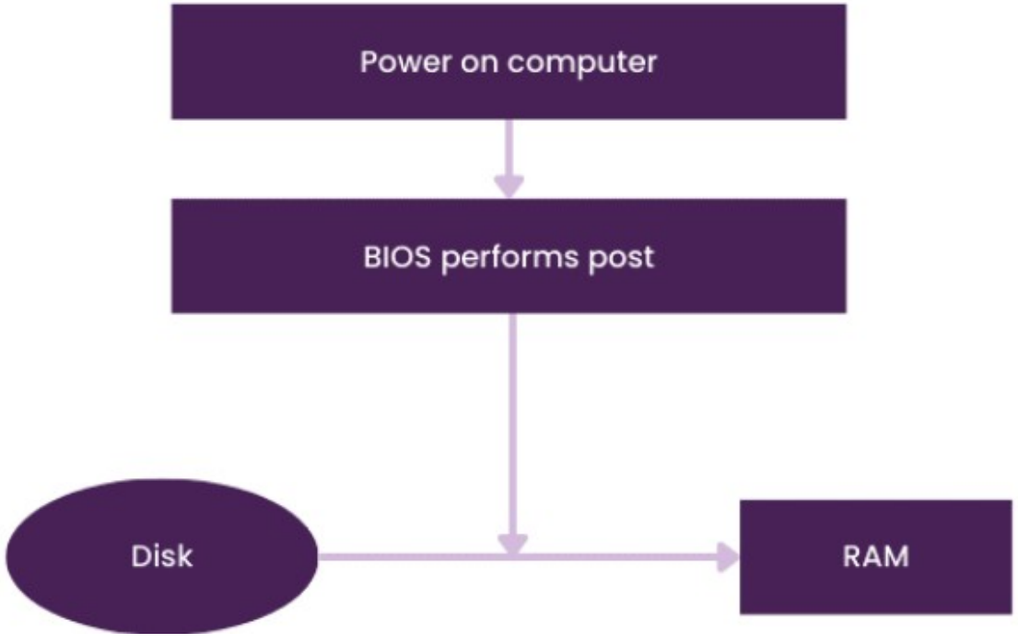


Fig 2. System Booting architecture

UPES
UNIVERSITY OF TOMORROW

System Booting: Process

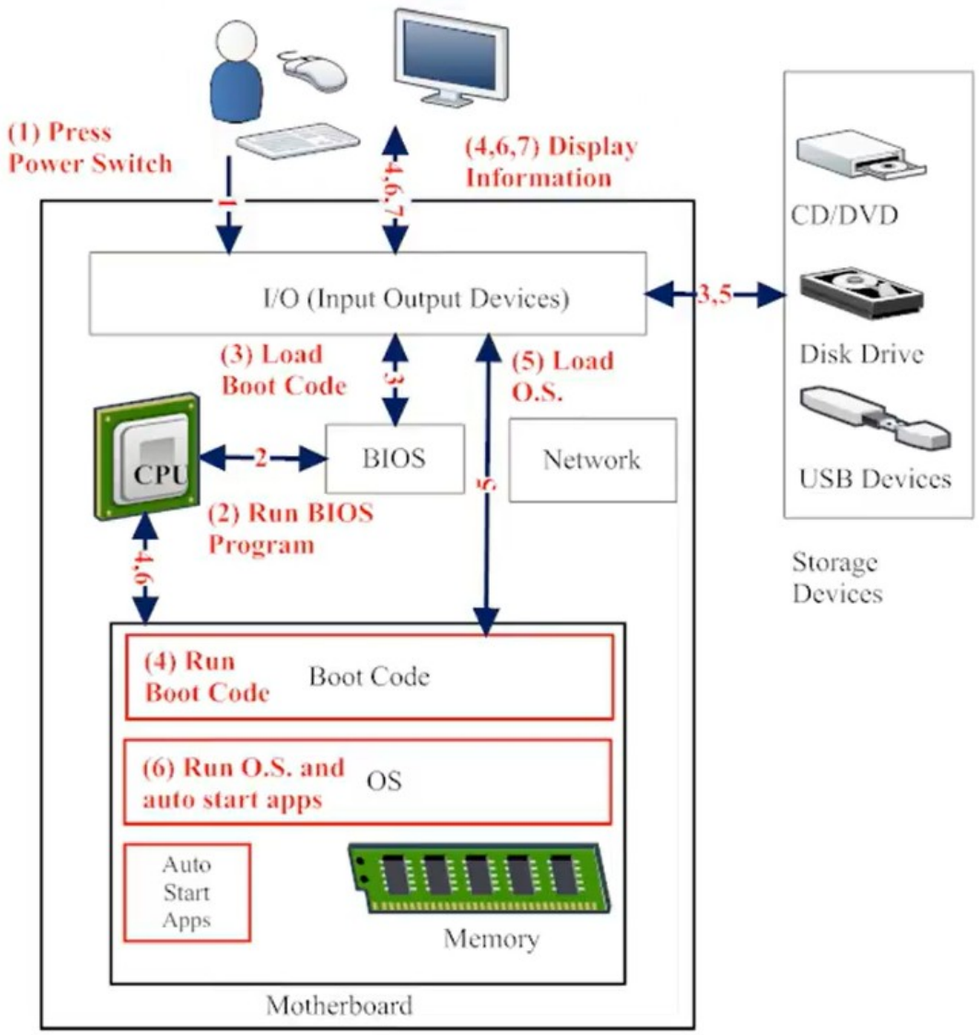


Fig 3. Components of Booting

The following components are involved during booting process

1. Power Supply
2. CPU
3. BIOS (ROM)
4. CMOS
5. Master Boot Record
6. RAM
7. Boot Loader
8. Storage Drives

UPES
UNIVERSITY OF TOMORROW

System Booting: Process

The booting process is as follows:

1. Power Supply

- The first step to switch on the power supply.
- This power supply provides electricity to all motherboard components like CPU, Hard disk etc.

2. CPU

- CPU loads the BIOS (Basic Input output System) to execute instructions stored in it.

3. BIOS

- When we turn on the computer, so the CPU looks for another program, called the BIOS (Basic Input/Output System), and runs it. The BIOS is a firmware that is located on the motherboard and is run by the CPU to start the booting sequence

Fig 4. ROM and BIOS Program

System Booting: Process

3. BIOS Initialization, RUN Test and Initialization Hardware

- BIOS settings are stored in a non-volatile memory called, CMOS (Complementary metal oxide semiconductor).
- BIOS loads the setting from CMOS
- CMOS is backed by CMOS battery to provide it continuous power supply even after system is shutdown or restart.
- Now BIOS program is loaded with setting received from CMOS
- BIOS program initialized POST test, called Power on Self-Test.
- POST contains some test cases for each hardware to ensure whether each Connected device working properly or not.
  - Printer, Keyboard, Mouse, Speaker, RAM etc.

Fig 5. CMOS battery

4. BIOS Sends Control to Boot Device

- Once the BIOS finished all initialization and hardware test. It will find the Boot Device to load operating system
- Now it looks for the actual program that will search and run the operating system
- Boot device contains the instruction for this actual program called boot loader.

System Booting

Boot Devices

- The boot device is the device from which the operating system is loaded. A modern **PC BIOS** (Basic Input/Output System) supports booting from various devices.
- These devices includes
    - Local hard disk drive,
    - Optical drive,
    - Floppy drive,
    - A network interface card,
    - A USB device.
- The BIOS will allow the user to configure a boot order. If the boot order is set to:
    - CD Drive
    - Hard Disk Drive
    - Network

The BIOS will try to boot from the CD drive first, and if that fails, then it will try to boot from the hard disk drive, and if that fails, then it will try to boot from the network, and if that fails, then it won't boot at all.

UPES
UNIVERSITY OF TOMORROW

System Booting

Boot Loader

- Boot Loader is a software program that is responsible for “actually loading” the operating system. The Boot Loader is typically a part of the Operating System itself.
- Till the point Boot Loader starts loading the OS, there is nothing in the Main Memory of the machine.
- After POST test, BIOS search for a place where bootloader is located.
- Boot loader is located at MBR (Master Boot Record) which further exists at 0^{th} index of boot disk
- BIOS instructs CPU to run bootloader in main memory (RAM) to start running OS

System Booting: Process

5. Boot loader initialize the OS

- Control is transferred from BIOS to CPU and CPU to boot loader
- Boot loader responsible to initialize the OS into main memory
- Boot loader is not a single program for all kind of OS
- Each operating system has its own instruction for initialization.
- Each OS manufacturer write its own program for bootloader
  - Microsoft has written bootmgr.exe bootloader to initialize Windows
  - Apple has written boot.efi bootloader to initialize Mac:
  - Linux has written Grub bootloader to initialize its OS
- The whole control is managed by bootloader
- Bootloader performs all the checks to initialize OS

System Booting: Types

Cold Booting

- When the computer starts for the first time or is in a shut-down state and switch on the power button to start the system
- This type of process to start the computer is called cold booting.
- During cold booting, the system will read all the instructions from the ROM (BIOS) and the Operating System will be automatically get loaded into the system.
- This booting takes more time than Hot or Warm Booting.

System Booting: Types

Warm Booting

- When computer systems come to no response or hang state, and then the system is allowed to restart during on condition.
- It is referred to as rebooting (Warm Booting).
- There are many reasons for this state, and the only solution is to reboot the computer. Rebooting may be required when we install new software or hardware or to set software or hardware configuration changes
- sometimes systems may behave abnormally or may not respond properly.
- In such a case, the system has to be a force restart.
- Most commonly **Ctrl+Alt+Del** button is used to reboot the system.

System Booting: Dual Booting

Dual booting is a configuration where two operating systems are installed on a single computer, allowing the user to select which OS to boot into at startup. This setup is useful for users who need to use features or applications specific to different operating systems.

Key Concepts

Boot Loader

- A boot loader is a small program that manages the boot process of a computer.
- It allows the user to choose between multiple operating systems at startup.
- Common boot loaders include GRUB (GNU GRand Unified Bootloader), LILO (Linux Loader), and Windows Boot Manager.

Partitioning

- Hard disk partitioning is crucial for dual booting. Each OS needs its own partition, and sometimes additional partitions are required for data or swap space.
- A typical dual-boot setup might include partitions for each OS and possibly a shared data partition accessible by both.

System Booting: Dual Booting

Key Concepts

File Systems

- Different operating systems often use different file systems. For example, Windows typically uses NTFS or FAT32, while Linux might use ext4, btrfs, or other file systems.
- Shared partitions should use a file system that both operating systems can read and write, such as FAT32 or exFAT.

UPES
UNIVERSITY OF TOMORROW

Dual Booting: Process

Step-by-Step Process

Backup Data

- Before making any changes to the disk, back up important data to prevent data loss.

Partition the Disk

- Use a partitioning tool (e.g., GParted, Disk Management in Windows) to create separate partitions for each operating system.
- Ensure there is enough space for each OS and its applications.

Install the First Operating System

- Install one of the operating systems first, if it's not already installed.
- During installation, allocate the appropriate partition for this OS.

UPES
UNIVERSITY OF TOMORROW

Dual Booting: Process

Install the Second Operating System

- Boot from the installation media of the second OS.
- During installation, select the partition designated for this OS. Be careful not to overwrite the existing OS.
- The installer of the second OS usually detects the presence of the first OS and configures the boot loader to allow choosing between the two.

Configure the Boot Loader

- If using GRUB, it will usually detect both operating systems and present a menu at startup.
- For Windows Boot Manager, you might need to add the second OS manually using tools like EasyBCD.

Post-Installation

- Update the OSes and drivers.
- Set up shared partitions or directories if needed.
- Customize the boot loader menu (e.g., setting default OS, timeout duration).

UPES
UNIVERSITY OF TOMORROW

Example: Dual Booting Windows and Linux

1. **Install Windows**
   - Install Windows first as it tends to overwrite existing boot loaders without asking.
   - During installation, create a partition for Windows, leaving space for Linux.

2. **Partitioning for Linux**
   - After installing Windows, boot from a Linux live CD/USB.
   - Use a partitioning tool to create partitions for Linux (e.g., root /, home /home, swap).

3. **Install Linux**
   - Install Linux on the newly created partitions.
   - The Linux installer will usually detect the Windows installation and add it to the GRUB boot menu.

4. **Boot Loader Configuration**
   - GRUB will be installed and set as the default boot loader.
   - After installation, you can edit the GRUB configuration to set the default OS and timeout.

UPES
UNIVERSITY OF TOMORROW

System Call

A system call is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.

In other words

A system call is a procedure, written in c, c++ or assembly level languages for performing various operations. A system call is a method of interacting with the operating system via programs. A system call is a request from computer software to an operating system's kernel.

System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

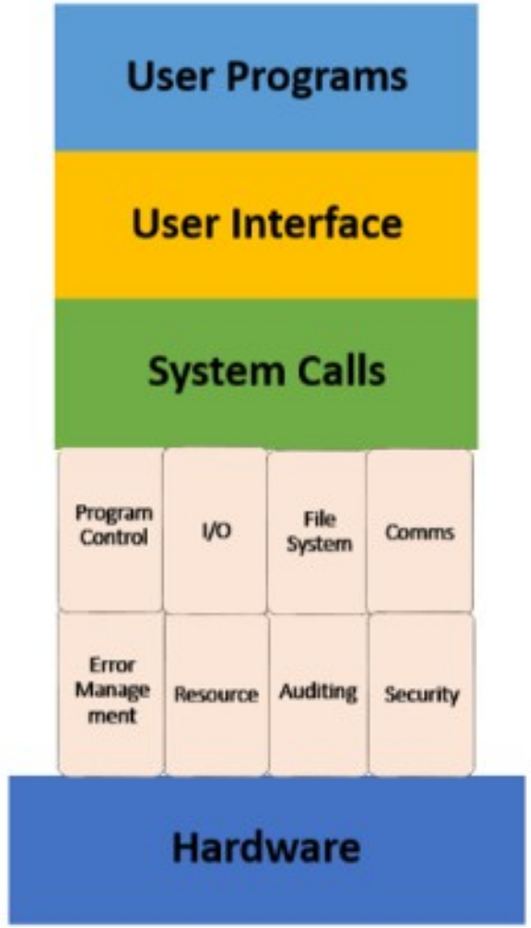


Fig 6. System call

UPES
UNIVERSITY OF TOMORROW

System Call: Several Definitions

- A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- It is a Programming interface to the services provided by the OS.
- It is typically written in a high-level language (C or C++)
- System call provides the services of the operating system to the user programs via Application Program Interface(API).
- It provides an interface between a process and an operating system to allow user-level processes to request services of the operating system.
- System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

System Call as Operating-System Operations

- Interrupt driven by hardware
- Software error or request creates exception or trap
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- Dual-mode operation allows OS to protect itself and other system components
  - User mode and kernel mode
  - Mode bit provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as privileged, only executable in kernel mode
    - **System call changes mode to kernel, return from call resets it to user**

Transition from User mode to Kernal Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate
  - program that exceeds allotted time

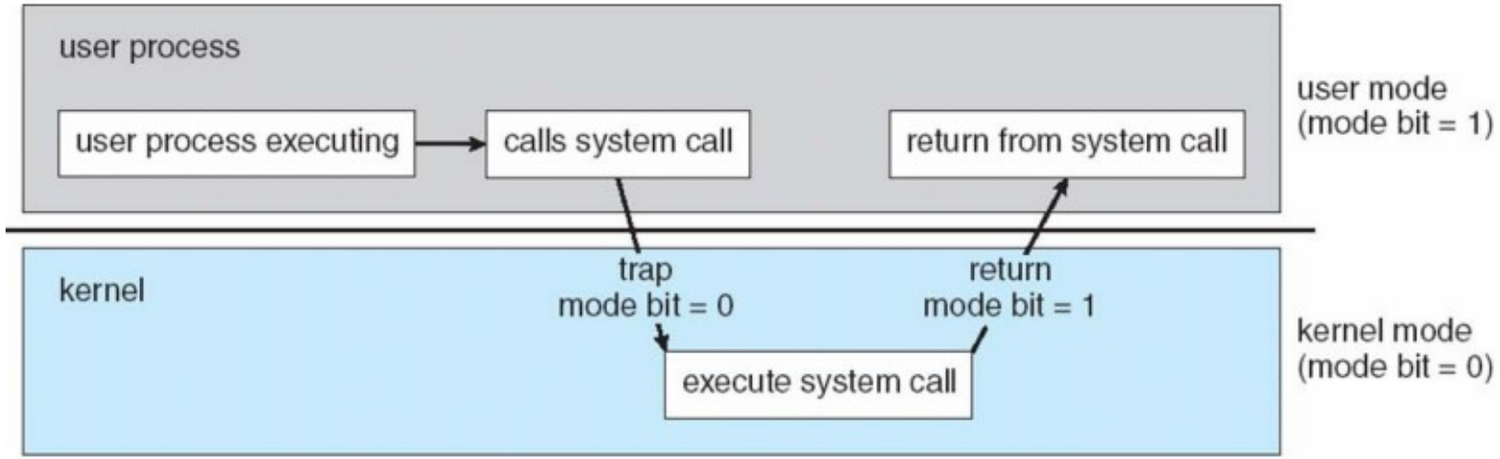


Fig 7. Transition from User mode to Kernal mode

System Call: Example

A System call sequence to copy the contents of one file to another file

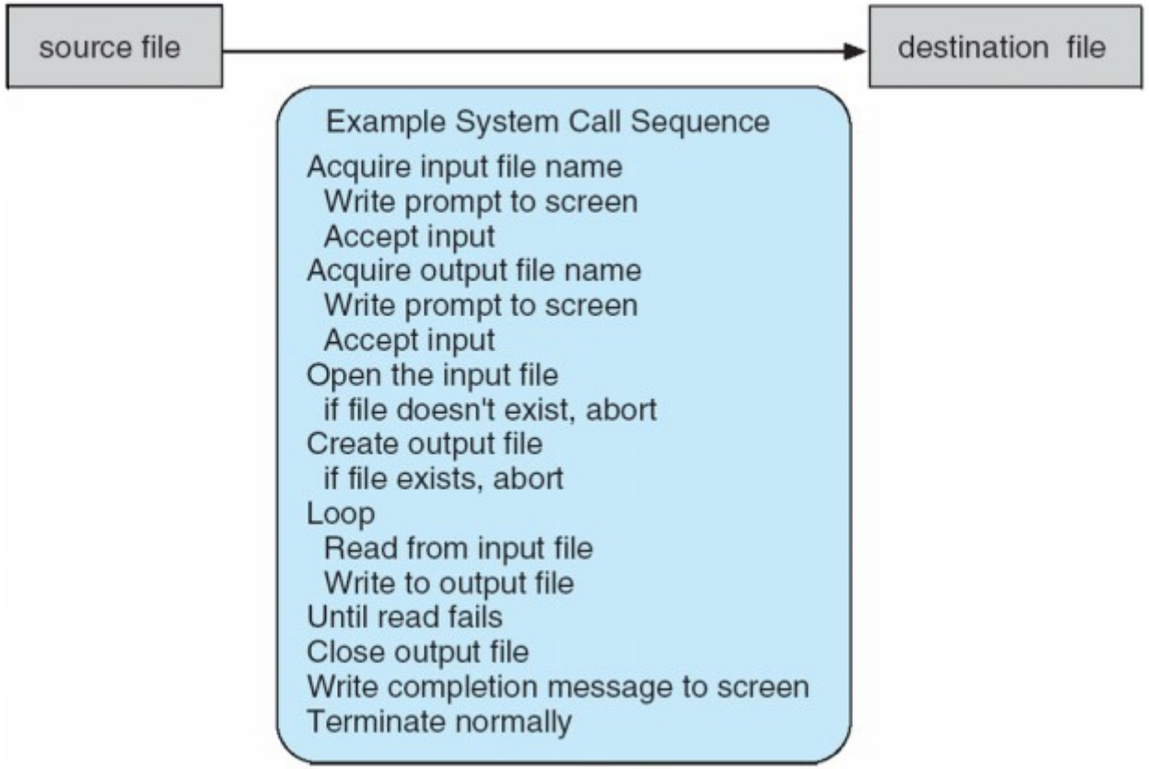
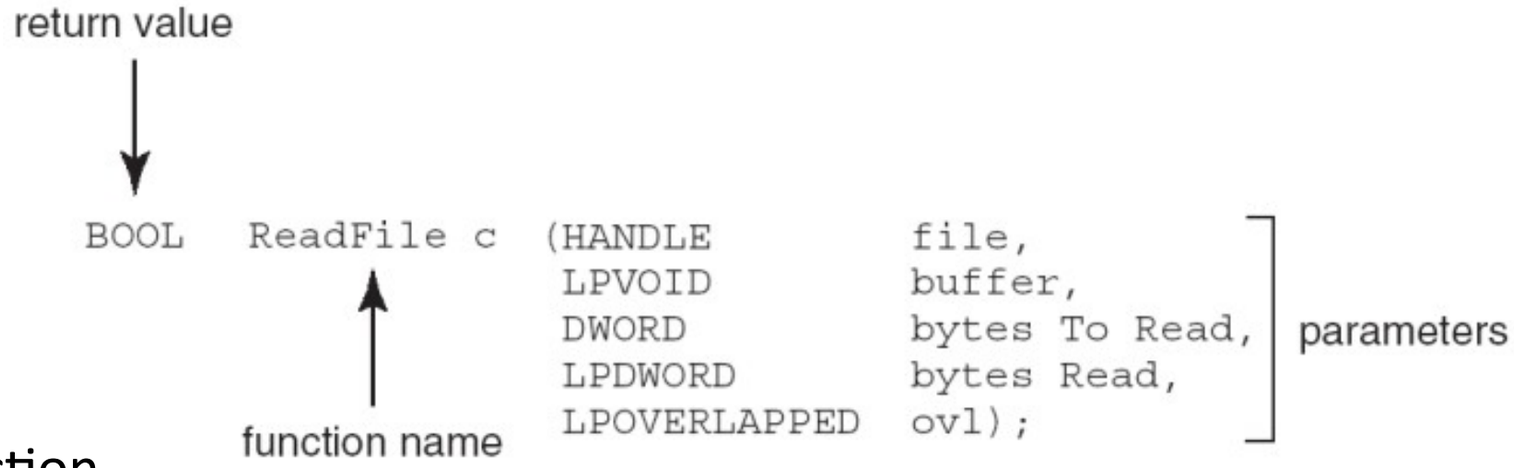


Fig 8. System call to copying one file to another

Standard API: Example

Consider the ReadFile() function in the Win32 API—a function for reading from a file



Parameters passed to ReadFile() function

- **HANDLE file**: The file to be read
- **LPVOID buffer:** A buffer where the data will be read into and written from
- **DWORD bytesToRead:** The number of bytes to be read into the buffer
- **LPDWORD bytesRead:** The number of bytes read during the last read
- **LPOVERLAPPED ovl: It** indicates if overlapped I/O is being used

System Call: Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API Managed by run-time support library (set of functions built into libraries included with compile)

Application Program Interface

- System call are complex procedures, hard to implements and it contents a numbers of parameters.
- Application programing Interface (API) are provided to simplify the concepts of system call.
- A programmer never use system call directly.
- Programmer uses API, which further uses system calls to implement services provided by OS
- An API is a set of protocols, routines, and tools for building software applications. It specifies how software components should interact, allowing different programs to communicate with each other

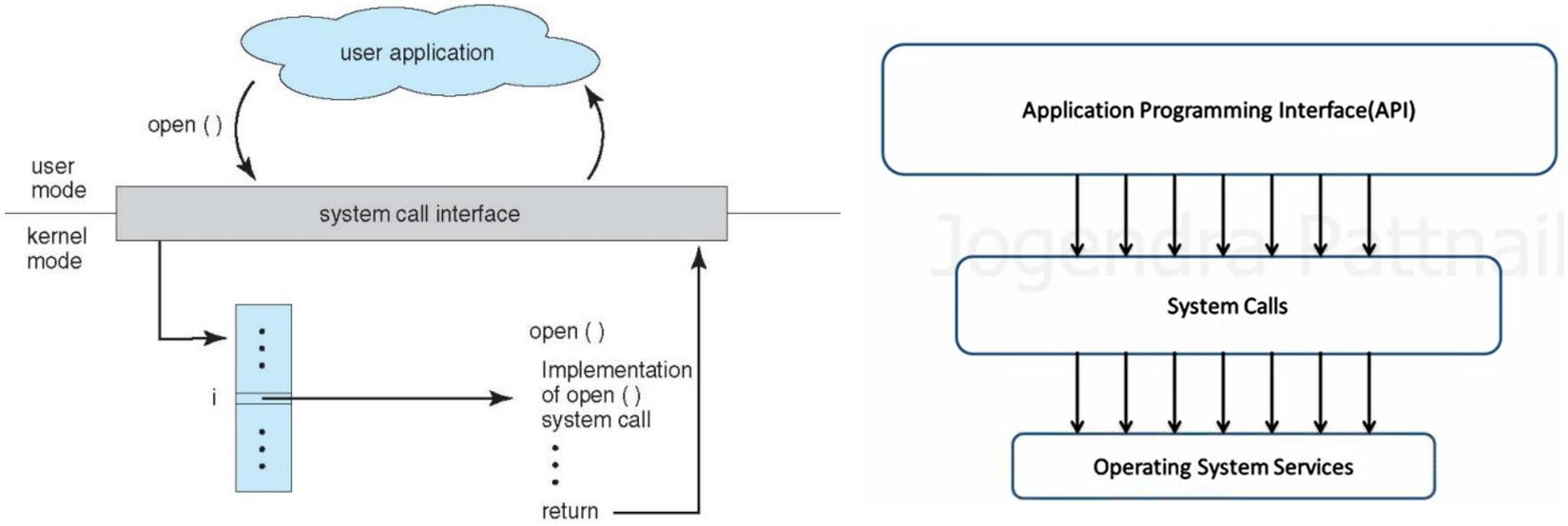


Fig 9. API - System call - OS Relationship

API – System Call - OS Relationship

Interaction Flow:

- **Application to API:** An application program makes a call to an API provided by the OS or a library.
- **API to System Call**: The API, which runs in user space, translates the high-level API function call into one or more system calls, which are the low-level interfaces to the OS.
- **System Call to OS Kernel:** The system call transfers control to the OS kernel, which then executes the corresponding kernel function to perform the requested operation.
- **OS Kernel to Hardware (if needed):**The kernel function may interact directly with hardware or manage resources such as memory, processes, or files.

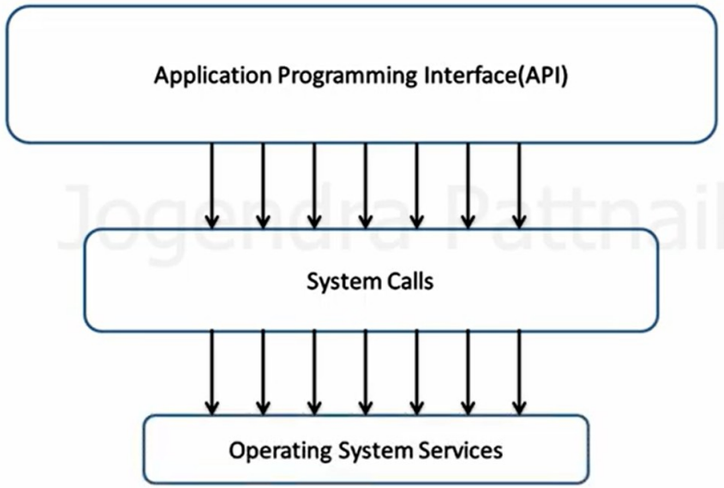


Fig 10. API - System call - OS Relationship

UPES
UNIVERSITY OF TOMORROW

API – System Call - OS Relationship: Example

Interaction Flow Example:

- Example: An application calls the **printf()** function from the C Standard Library (API).
- **printf()** internally calls the write() system call.
- The **write()** system call invokes the kernel function to write data to the appropriate file descriptor (e.g., a terminal or a file).
- The OS kernel handles the hardware interaction (if necessary) and completes the request.

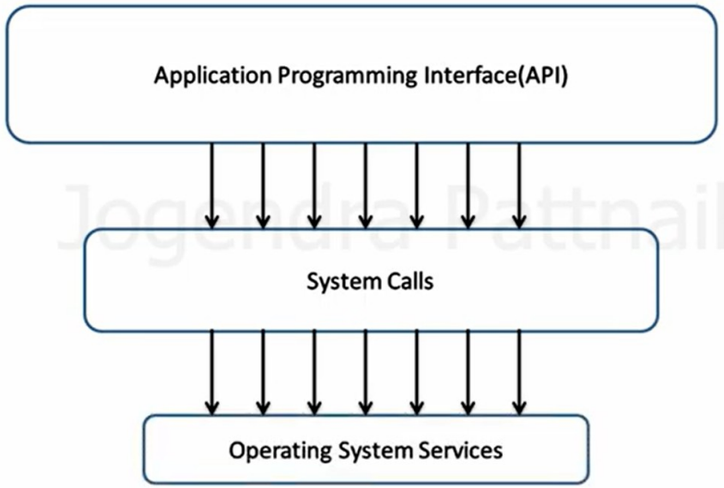


Fig 10. API - System call - OS Relationship

Application Program Interface

- Some of the popular API are as follows:

- **POSIX API:** This API works with UNIX, LINUX and MacOS X
- **Win32 API:** This API works with Windows
- **Java API:** This API works with Java Virtual Machine

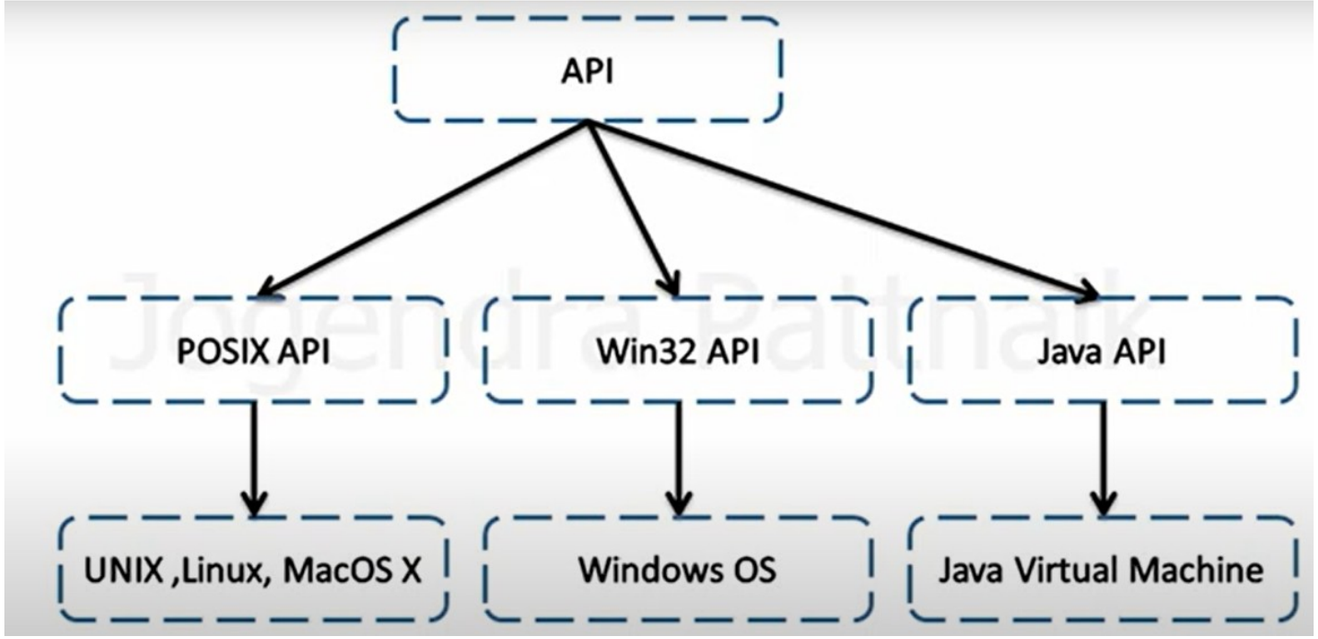


Fig 11. API and Designated OS

System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  1. Simplest: pass the parameters in registers
     - In some cases, may be more parameters than registers
  2. Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
     - This approach taken by Linux and Solaris
  3. Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table

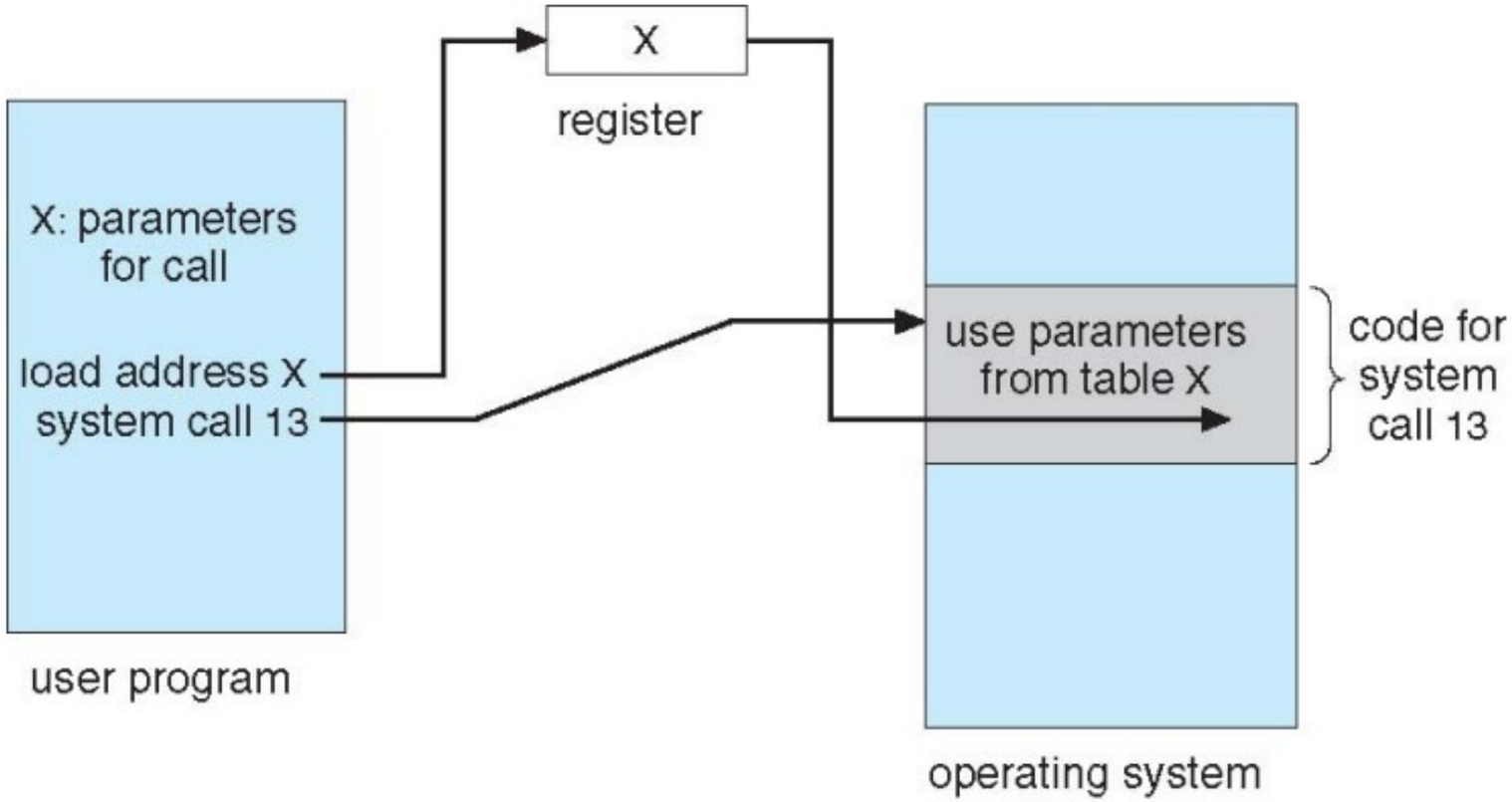


Fig 12. Parameter passing using Table

System Call Table

A system call table is a data structure used by an operating system (OS) kernel to manage and dispatch system calls. It maps system call numbers, which are unique identifiers for each system call, to the corresponding kernel functions that implement these calls.

Purpose:

- To provide a structured and efficient mechanism for the OS to handle requests from user applications.
- To ensure that system calls are processed securely and reliably by the kernel.

System Call Table

Structure and Components

System Call Numbers

- Each system call is assigned a unique number.
- This number is used as an index to access the corresponding function in the system call table.

Function Pointers

- The table consists of function pointers, where each pointer points to a specific kernel function that implements a system call.
- The functions are written to handle various operations such as file handling, process control, memory management, etc.

```
0     common  read            sys_read
1     common  write           sys_write
2     common  open            sys_open
3     common  close           sys_close
```

Fig 13. System call Table (syscall_64.tbl) with system call number, its name, and the corresponding kernel function

System Call Table: Working

Invocation

- An application invokes a system call via a library function (e.g., the read function from the C Standard Library).
- The library function uses a special instruction to switch from user mode to kernel mode, passing the system call number and its arguments.

Kernel Mode

- The CPU switches to kernel mode and the system call dispatcher function in the kernel is executed.
- The dispatcher uses the system call number to index into the system call table.

Dispatch

- The corresponding function pointer is retrieved from the table.
- The kernel calls the function to perform the requested operation.

Completion

- The system call handler completes the operation and returns a result.
- Control is returned to the user mode application with the result of the system call.

System Call: How it gets execute

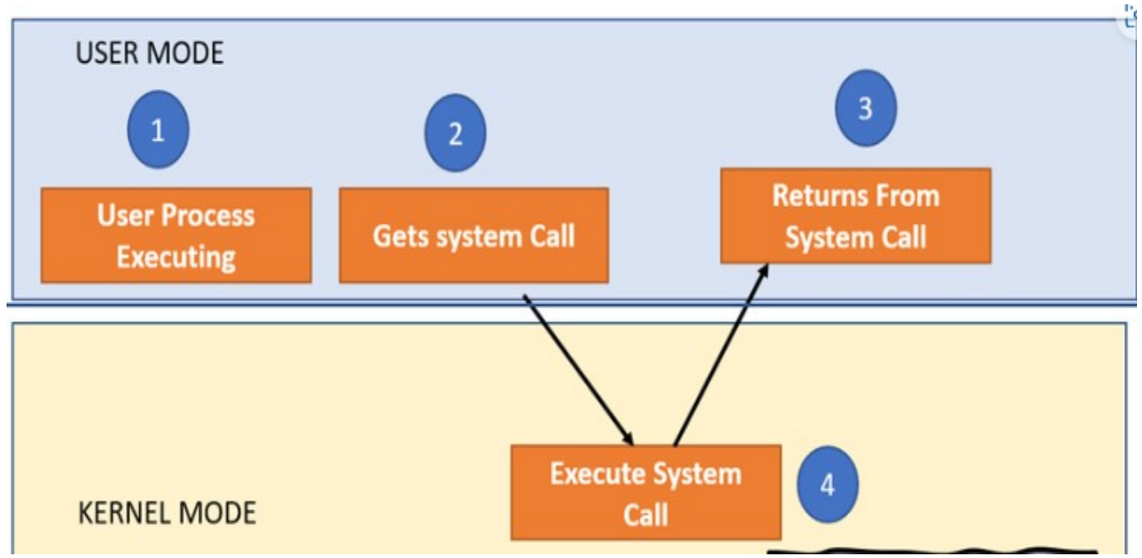


Fig 14. System Call Execution

Step1: The processes executed in the user mode till the time a system call interrupts it.

Step 2: Subsequently, the system call is executed in the kernel-mode on a priority basis.

Step 3: Once system call execution is over, control returns to the user mode.

Step 4: The execution of user processes resumed in Kernel mode.

UPES
UNIVERSITY OF TOMORROW

System Call: Necessity

There are some following scenario where system calls are required in OS

- A system call is required if a user want to perform read and write operations with files demand
- If a file system wants to create or delete files, system calls are required.
- System calls are used for the creation and management of new processes.
- Network connections need system calls for sending and receiving packets.
- Access to hardware devices like scanner, printer, need a system call.

System Call: Types

There are five types of System Calls in OS

Fig 15. System call Types

System Call: Types

PROCESS CONTROL

This system calls perform the task of process creation, process termination, etc. **Functions**

- End and Abort a process
- Load and Execute process
- Create Process and Terminate Process
- Wait and Signal Event
- Allocate and free memory

FILE MANAGEMENT

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

Functions

Create a file, Delete file, Open and close file, Read, write, and reposition
Get and set file attributes

UPES
UNIVERSITY OF TOMORROW

System Call: Types

DEVICE MANAGEMENT

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.

Functions

- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

INFORMATION MAINTENANCE

It handles information and its transfer between the OS and the user program.

Functions

- Get or set time and date
- Get process and device attributes

UPES
UNIVERSITY OF TOMORROW

System Call: Types

COMMUNICATIONS

These types of system calls are specially used for inter-process communications.

Functions

- Create, delete communications connections
- Send, receive message
- Help OS to transfer status information
- Attach or detach remote devices

UPES
UNIVERSITY OF TOMORROW

System Call: Windows & Unix

Process	Windows	Unix
Process Control	CreateProcess(), ExitProcess(), WaitForSingleObject()	Fork(), Exit(), Wait()
File Manipulation	CreateFile(), ReadFile(), WriteFile(), CloseHandle()	Open(), Read(), Write(), Close()
Device Management	SetConsoleMode(), ReadConsole(), WriteConsole()	Ioctl(), Read(), Write()
Information Maintenance	GetCurrentProcessID(), SetTimer(), Sleep()	Getpid(), Alarm(), Sleep()
Communication	CreatePipe(), CreateFileMapping(), MapViewOfFile()	Pipe(), Shmget(), Mmap()
Protection	SetFileSecurity(), InitializeSecurityDescriptor(), SetSecurityDescriptorgroup()	Chmod(), Umask(), Chown()

System Call: Windows & Unix

	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()

Fig 16. System Call, Windows vs Linux

UPES
UNIVERSITY OF TOMORROW

System Call: Some System Calls

System Call	Explanation	System Call	Explanation
open()	It allows you to access a file on a file system	**Exec()**	When an executable file replaces an earlier executable file in an already executing process, this system function is invoked.
read()	It is used to obtain data from a file on the file system	**Close()**	It is used to end file system access. When this system call is invoked, it signifies that the program no longer requires the file, and the buffers are flushed
Wait()	It is used to suspend the parent process. Once the child process has completed its execution, control is returned to the parent process.	**Write ()**	It is used to write data from a user buffer to a device like a file. This system call is one way for a program to generate data
fork()	It is used to generate child process	**Exit()**	It is used to end program execution.

UPES
UNIVERSITY OF TOMORROW

Open-Source Operating System

- The term "**open source**" refers to computer software or applications where the owners or copyright holders enable the users or third parties to use, see, and edit the product's source code
- The **open-source operating system** allows the use of code that is freely distributed and available to anyone and for commercial purposes.
- Being an open-source application or program, the program source code of an open-source OS is available. The user may modify or change those codes and develop new applications according to the user requirement.

Open-Source Operating Systems: Working

- It works similarly to a closed operating system, except that the user may modify the source code of the program or application. There may be a difference in function even if there is no difference in performance.

- For instance, the information is packed and stored in a proprietary (closed) operating system. In open-source, the same thing happens. However, because the source code is visible to you, you may better understand the process and change how data is processed.

- Some Open-Source Operating Systems are as follows:

Linux Kernal	**Linux Lite**	**Linux Mint**	**Fedora**
Solus	**Chrome OS**	**React OS**	

UPES
UNIVERSITY OF TOMORROW

MCQ

Q1. Which component performs the POST (Power-On Self-Test)?

A. CPU
B. BIOS
C. RAM
D. Bootloader

Q2. Where are BIOS settings stored?

A. RAM
B. Hard Disk
C. CMOS
D. SSD

Q3. Where is the Master Boot Record (MBR) located?

A. At the beginning of the RAM
B. At the beginning of the hard disk
C. In the BIOS
D. In the CMOS

Q4. What is a cold boot?

A. Restarting the system without powering off
B. Starting the system from a powered-off state
C. Updating the BIOS settings
D. Entering the boot menu

UPES
UNIVERSITY OF TOMORROW

MCQ

Q5. What is dual booting?

A. Running two operating systems simultaneously

B. Installing two operating systems on a single computer

C. Loading the operating system twice

D. Booting from two different storage devices

Q6. What is a system call?

A. A hardware function

B. A software program that manages computer hardware

C. Interface between a process and the operating system

D. A type of virus

Q7. Which of the following is a popular API used with UNIX, LINUX, and MacOS X?

A. Win32 API

B. Java API

C. POSIX API

D. Python API

Q8. What does the read() system call do?

A. Writes data to a file

B. Reads data from a file

C. Closes a file

D. Opens a file

UPES
UNIVERSITY OF TOMORROW

MCQ

Question No-Answer	Option	Description
Q1-Answer	B	BIOS
Q2-Answer	C	CMOS
Q3-Answer	B	At the beginning of the hard disk
Q4-Answer	B	Starting the system from a powered-off state
Q5-Answer	B	Installing two operating systems on a single computer
Q6-Answer	C	interface between a process and the operating system
Q7-Answer	C	POSIX API
Q8-Answer	B	Reads data from a file

Summary/Key Points

- Booting is a start-up sequence that starts the operating system of a computer when it is turned on.
- Booting sequence comprises of three components: Boot Loader, Boot device & Boot sequence
- Booting is of two types: Cold booting and Warm booting
- A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- Process control system calls perform the task of process creation, process termination, etc.
- File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.
- Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.
- Information maintenance system call handles information and its transfer between the OS and the user program.
- Communication system calls are specially used for inter-process communications
- The open-source operating system allows the use of code that is freely distributed and available to anyone and for commercial purposes.

UPES
UNIVERSITY OF TOMORROW

Reference Material

- Operating Systems Concepts (10th Ed.) Silberschatz A, Peterson J and Galvin P, John Wiley & Sons, Inc. 2018.

  Topic:
  - Open-Source Operating Systems, Page No: 46-47
  - System Calls, Page No: 62-65
  - Types of System Calls, Page No: 66-73
  - System boot, Page No: Page No: 94-95

- Modern Operating Systems (4th Ed.) by Andrew S. Tanenbaum and Herbert Bos, 2007
- Operating Systems: Principles and Practice by Thomas Anderson and Michael Dahlin, 2014

UPES
UNIVERSITY OF TOMORROW

Coming Up-Next Lecture

- Process: Program and Process concept
- Process in memory
- Process Control Block
- Process States

UPES
UNIVERSITY OF TOMORROW

Thank You

UPES
UNIVERSITY OF TOMORROW