1.3 OS INTERFACES: SYSTEM CALLS, SHELL

In an operating system (OS), the shell and system calls function as essential interfaces that bridge the gap between users, application programs, and the core of the system (the kernel).

1. SHELL

1. The Shell (User Interface)

- The shell is the outermost layer of the OS that serves as a **direct interface** for human users.
- It interprets user commands and translates them into a format the OS kernel can understand.

2. Primary Types:

  - **Command-Line Interface (CLI):** Text-based environments like Bash (Linux/macOS), PowerShell, or Command Prompt (Windows).
  - **Graphical User Interface (GUI):** Visual environments using icons and menus, such as GNOME or Windows Explorer.

3. Core Functions:

- It parses inputs, executes programs, manages input/output redirection (e.g., using > or |), and supports shell scripting for task automation

OS Interface Hierarchy Diagram

The architecture is typically represented as concentric circles or vertical layers:

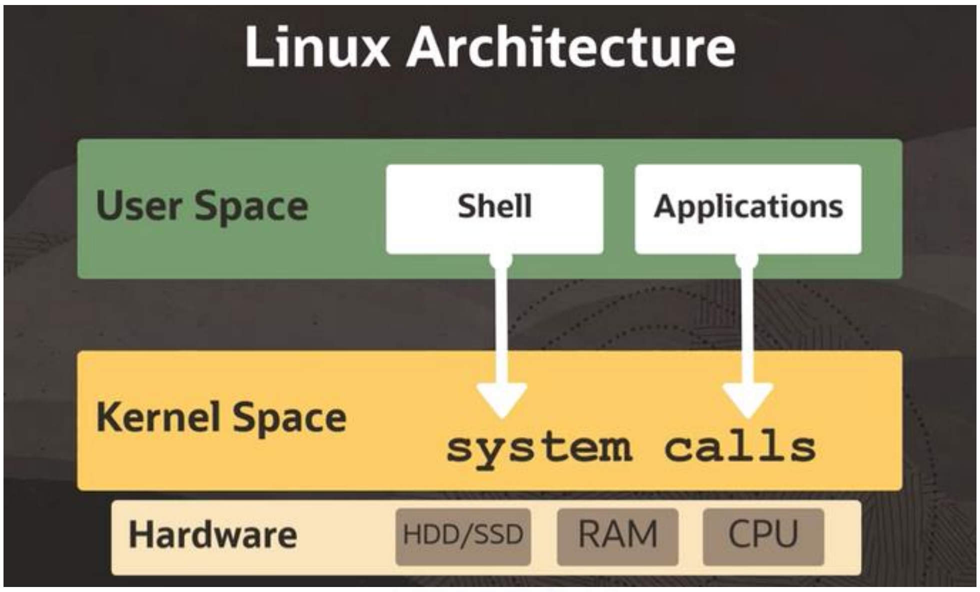
Users: Interact directly with the shell via commands or graphical icons.

Shell (The Interface): Translates user input into instructions the system can execute.

- **CLI (Command-Line):** Text-based (e.g., Bash, PowerShell).
- **GUI (Graphical):** Visual (e.g., Windows Explorer, GNOME).
- **Voice based Interface:** Voice (e.g., Siri, Alexa)

Kernel (The Core): The central part of the OS that manages resources and talks to hardware.

Hardware: The physical components (CPU, RAM, Disk) managed by the kernel.



How the Shell Works

Read: The shell waits for user input (a command like ls or a mouse click).

Evaluate: It parses the input to determine which program or service is required.

Execute: It invokes system calls to request the kernel to perform tasks like opening a file or printing to the screen.

Output: Once the kernel completes the task, the shell displays the results back to the user.

Example:

- When a user opens a file in Windows, the GUI Shell sends a request to the Kernel, which retrieves the file and displays it.

In simple terms, Kernel is the heart of the OS, managing resources and hardware whereas the Shell is the user interface, allowing interaction with the system.

2. SYSTEM CALLS IN OPERATING SYSTEM

- A system call is an interface between a **program** running in user space and the **operating system** (OS).
- Application programs use system calls **to request services** and functionalities from the OS's kernel.
- When a program invokes a system call, the execution **context switches** from user to kernel mode, allowing the system to access hardware and perform the required operations safely.

- After the operation is completed, the control returns to user mode, and the program continues its execution.

Advantages

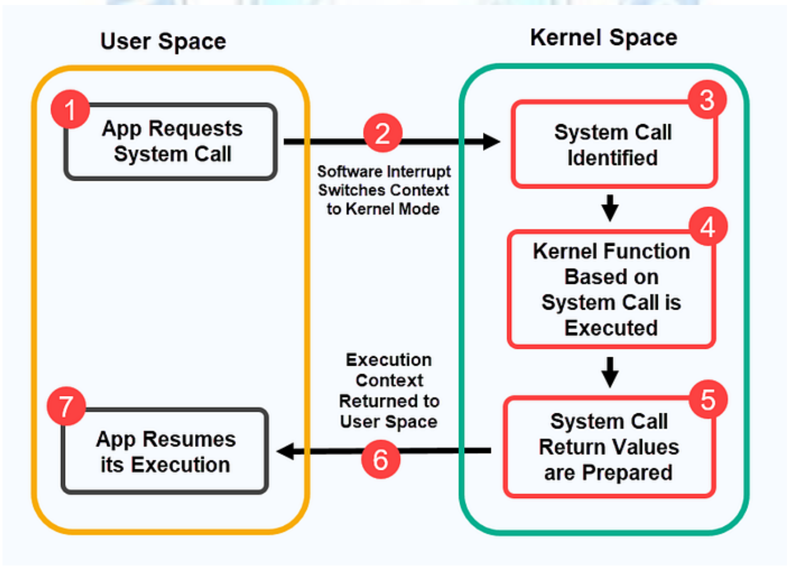
- Ensures that hardware resources are isolated from user space processes.
- Prevents direct access to the kernel or hardware memory.
- Allows application code to run across different hardware architectures.

How Do System Calls Work?

1. **System Call Request**. The application requests a system call by invoking its corresponding function. For instance, the program might use the read() function to read data from a file.

2. **Context Switch to Kernel Space**. A software interrupt or special instruction is used to trigger a context switch and transition from the user mode to the kernel mode.

3. **System Call Identified**. The system uses an index to identify the system call and address the corresponding kernel function.


User Space
Kernel Space
1
App Requests System Call
2
Software Interrupt Switches Context to Kernel Mode
3
System Call Identified
4
Kernel Function Based on System Call is Executed
5
System Call Return Values are Prepared
6
Execution Context Returned to User Space
7
App Resumes its Execution

5. **System Prepares Return Values**. After the kernel function completes its operation, any return values or results are prepared for the user application.

6. **Context Switch to User Space**. The execution context is switched back from kernel mode to user mode.

7. **Resume Application**. The application resumes its execution from where it left off, now with the results or effects of the system call.

Features of System Calls

- Security
- Abstraction
- Access control
- Consistency

Types of System Calls

1. Process Control

System calls play an essential role in controlling system processes. They enable you to:

- Create new processes or terminate existing ones.
- Load and execute programs within a process's space.
- Schedule processes and set execution attributes, such as priority.
- Wait for a process to complete or signal upon its completion.

Purpose	**System Call**
Create a new process	`fork()`
Execute a new program	`exec()`
Terminate process	`exit()`
Wait for a child process	`wait()`
Get process ID	`getpid()`
Change priority	`nice()`

Exec System Calls (Program Execution)

Exec system calls replace the current process image with a new program.

Common exec family calls:

- execl()
- execv()
- execvp()
- execve()

Example Flow

fork() → creates child

exec() → loads new program into child

2. File Management

System calls support a wide array of file operations, such as:

- Reading from or writing to files.
- Opening and closing files.
- Deleting or modifying file attributes.
- Moving or renaming files.
- Press enter or click to view image in full size

Operation	System Call Example
Create a file	`creat()` / `open()`
Open a file	`open()`
Read from file	`read()`
Write to file	`write()`

Linux/Unix Function	Description
mkdir()	Create a new directory.
rmdir()	Remove a directory.

3. Device Management

System calls can be used to facilitate device management by:

- Requesting device access and releasing it after use.
- Setting device attributes or parameters.
- Reading from or writing to devices.
- Mapping logical device names to physical devices.

Linux/Unix Function	Description
brk() or sbrk()	Increase or decrease the program’s data space.
mmap()	Map files or devices into memory.

4. Information Maintenance

This type of system calls enables processes to:

- Retrieve or modify various system attributes.
- Set the system date and time.
- Query system performance metrics.

Linux/Unix Function	Description
time()	Get the current time.
getuid()	Get the user ID.
getgid()	Get the group ID.

5. Communication

The communication call type facilitates:

- Sending or receiving messages between processes.
- Synchronizing actions between user processes.
- Establishing shared memory regions for inter-process communication.
- Networking via sockets.

Linux/Unix Function	Description
socket()	Create a new socket.
bind()	Bind a socket to a network address.
listen()	Listen for connections on a socket.
accept()	Accept a new connection on a socket.
connect()	Initiate a connection on a socket.
send() or recv()	Send and receive data on a socket.

6. Security and Access Control

System calls contribute to security and access control by:

- Determining which processes or users get access to specific resources and who can read, write, and execute resources.
- Facilitating user authentication procedures.

Linux/Unix Function	Description
chmod() or umask()	Change the permissions/mode of a file.
chown()	Change the owner and group of a file.

User types a command in the shell → shell parses it → shell makes a system call → kernel executes the request → kernel returns the result → shell displays it to the user.

3. Terminal

A terminal is a text-based interface that allows a user to interact with the operating system using commands instead of graphical icons.

It acts as a bridge between the user and the OS kernel through a shell (command interpreter).

Component	**Description**
Terminal	The screen or window where you type commands and see the output.
Shell	The command-line interpreter that processes the commands (e.g., *bash, zsh, sh*).
Kernel	The core part of the OS that executes the requested operations.
System Calls	Low-level functions used by the shell and commands to communicate with the kernel.

Working of Terminal Interface

1. User enters a command → e.g., ls -l
2. Shell interprets it and translates it into a system call (like open(), read(), write()).
3. Kernel executes the request (e.g., read directory contents).
4. Output is displayed back on the terminal.

4. Linux Shell

- A shell in Linux is a command-line interface (CLI) that allows users to interact with the operating system by typing commands.
- It acts as an interpreter between the user and the OS kernel — converting human-readable commands into system calls that the kernel can understand and execute.

Position of Shell in OS Architecture

USER
SHELL (BASH, SH)
KERNEL (LINUX CORE)
HARDWARE RESOURCES

Main Functions of the Shell

Function	Description
Command Interpretation	Reads and executes commands typed by the user.
Program Execution	Runs programs or scripts (e.g., ls, cat, gcc, etc.).
Input/Output Redirection	Controls data flow between files and commands (>, <, >>).
Pipelines	Connects commands using `
Environment Control	Manages environment variables (PATH, HOME, etc.).
Scripting	Automates tasks using shell scripts (.sh files).

Types of Linux Shells

Shell Type	Description
Bash (Bourne Again Shell)	Most common and default shell in Linux.
sh (Bourne Shell)	The original Unix shell, simple and stable.
csh / tcsh	C-like syntax, suitable for programmers.
ksh (Korn Shell)	Combines features of Bourne and C shells.
zsh (Z Shell)	Advanced shell with auto-correction, themes, and plugins.
fish (Friendly Interactive Shell)	User-friendly, with syntax highlighting.

Examples of Linux Shell Commands

Command	Description
pwd	Print current working directory.
ls -l	List files in detailed format.
cd /home	Change directory.
mkdir test	Create a new directory.
rm file.txt	Remove a file.
cat file.txt	Display file contents.
grep "word" file.txt	Search for a word in a file.
echo "Hello"	Display text on the screen.
chmod 755 file	Change file permissions.
ps	Display running processes.