**Structure of Linux Operating System**

The Linux Operating System follows a modular design and layered architecture, making it highly flexible and robust. Each layer has specific functions and interacts with other layers to provide a seamless computing environment. Below is the detailed structure of the Linux operating system:

**1. Hardware Layer**

This is the bottom-most layer of the Linux architecture, consisting of physical components such as:

* CPU
* Memory (RAM)
* Hard disks and SSDs
* Input/output devices (keyboard, mouse, etc.)
* Networking hardware (Ethernet, Wi-Fi cards)

The hardware layer interacts directly with the Linux kernel through device drivers.

**2. Kernel**

The kernel is the core of the Linux operating system. It acts as a bridge between the hardware and software layers. The kernel manages system resources and facilitates communication between hardware and software.

**Functions of the Kernel:**

* **Process Management**: Handles process creation, scheduling, and termination.
* **Memory Management**: Allocates and manages memory (RAM) for processes.
* **Device Drivers**: Interfaces with hardware devices (e.g., disk drives, network interfaces).
* **File System Management**: Manages file operations and storage systems.
* **System Calls**: Provides an interface for applications to request kernel services.

**Types of Kernels:**

* **Monolithic Kernel**: Used in Linux; all functionalities (e.g., drivers, file systems) are included in a single kernel binary.
* **Microkernel**: Opposite of monolithic, used in some other operating systems.

**3. Shell**

The shell is the command-line interpreter (CLI) that provides a user interface to interact with the kernel. It interprets user commands and passes them to the kernel for execution.

**Types of Shells in Linux:**

* **Bourne-Again Shell (Bash)**: Default shell in most Linux distributions.
* **Z Shell (Zsh)**: Advanced shell with additional features.
* **Korn Shell (ksh)**: Known for scripting capabilities.
* **Fish Shell**: User-friendly and interactive.

**Shell Functions:**

* Command execution
* Scripting (e.g., automating tasks through shell scripts)
* Customization (e.g., aliases, environment variables)

**4. System Libraries**

System libraries provide standard functions that allow user programs to interact with the kernel without needing to implement low-level system calls directly. These libraries simplify application development.

**Examples:**

* **GNU C Library (glibc)**: Commonly used for standard input/output, memory allocation, and string operations.
* **Dynamic Linking Libraries (DLLs)**: Shared libraries loaded during program execution.

**5. System Utilities**

System utilities are essential tools and programs used for system management and maintenance. These are divided into:

* **Basic Utilities**: File operations, disk management, and text processing.
* **Administrative Utilities**: User management, software installation, and system configuration.

**Examples:**

* ls, cat, grep (basic commands)
* systemctl, useradd, apt or yum (administrative commands)

**6. User Applications**

At the top layer, user applications are software programs used by end users. These programs leverage system libraries and kernel functions for execution.

**Examples:**

* **Graphical Applications**: Browsers (e.g., Firefox), office suites (e.g., LibreOffice), media players.
* **Development Tools**: Compilers (e.g., GCC), editors (e.g., Vim, Emacs).
* **Server Applications**: Web servers (e.g., Apache, Nginx), databases (e.g., MySQL, PostgreSQL).

**Linux System Architecture Diagram**

+—————————+

|        User Applications  |

+—————————+

|        System Utilities   |

+—————————+

|        System Libraries   |

+—————————+

|           Shell           |

+—————————+

|          Kernel           |

|  – Process Management     |

|  – Memory Management      |

|  – Device Drivers         |

|  – File System Management |

|  – Network Stack          |

+—————————+

|          Hardware         |

+—————————+

**Interaction Between Layers**

1. **Hardware ↔ Kernel**:
   1. Device drivers in the kernel interact directly with the hardware components to manage their functions.
2. **Kernel ↔ Shell/System Libraries**:
   1. The kernel provides low-level system calls that are accessed via system libraries or shell commands.
3. **Shell ↔ User Applications**:
   1. The shell interprets user commands and executes them by interacting with the kernel or utilities.
4. **User Applications ↔ System Libraries**:
   1. Applications use system libraries to perform high-level tasks like file handling, networking, and graphics rendering.

**Advantages of Linux Architecture**

1. **Modular Design**:
   1. Each layer is independent, allowing flexibility and easier debugging.
2. **Portability**:
   1. The modularity makes Linux adaptable to different hardware platforms.
3. **Security**:
   1. The kernel and shell enforce strict permissions and user isolation.
4. **Customization**:
   1. Users can modify or replace layers (e.g., use different shells or window managers).
5. **Efficiency**:
   1. Efficient resource management and low overhead make Linux suitable for high-performance systems.

**Conclusion**

The Linux operating system’s layered and modular structure is key to its flexibility, reliability, and widespread use. Each layer performs a specific role, contributing to a cohesive system capable of handling a vast range of applications, from personal computing to enterprise-grade servers and embedded systems.

Given that the goal of this article is to introduce you to the Linux kernel and explore its architecture and major components, let's start with a short tour of Linux kernel history, then look at the Linux kernel architecture from 30,000 feet, and, finally, examine its major subsystems. The Linux kernel is over six million lines of code, so this introduction is not exhaustive. Use the pointers to more content to dig in further.

A short tour of Linux history

**Linux or GNU/Linux?**

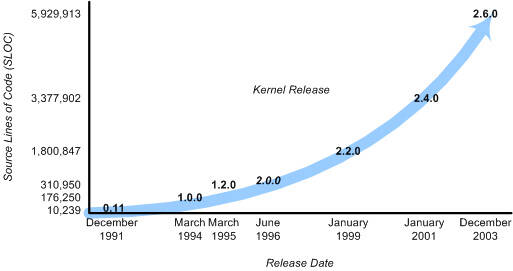
You've probably noticed that Linux as an operating system is referred to in some cases as "Linux" and in others as "GNU/Linux." The reason behind this is that Linux is the *kernel* of an operating system. The wide range of applications that make the operating system useful are the *GNU software*. For example, the windowing system, compiler, variety of shells, development tools, editors, utilities, and other applications exist outside of the *kernel*, many of which are *GNU software*. For this reason, many consider "GNU/Linux" a more appropriate name for the operating system, while "Linux" is appropriate when referring to just the *kernel*.

While Linux is arguably the most popular open source operating system, its history is actually quite short considering the timeline of operating systems. In the early days of computing, programmers developed on the bare hardware in the hardware's language. The lack of an operating system meant that only one application (and one user) could use the large and expensive device at a time. Early operating systems were developed in the 1950s to provide a simpler development experience. Examples include the General Motors Operating System (GMOS) developed for the IBM 701 and the FORTRAN Monitor System (FMS) developed by North American Aviation for the IBM 709.

In the 1960s, Massachusetts Institute of Technology (MIT) and a host of companies developed an experimental operating system called Multics (or Multiplexed Information and Computing Service) for the GE-645. One of the developers of this operating system, AT&T, dropped out of Multics and developed their own operating system in 1970 called Unics. Along with this operating system was the C language, for which C was developed and then rewritten to make operating system development portable.

Twenty years later, Andrew Tanenbaum created a microkernel version of UNIX®, called MINIX (for minimal UNIX), that ran on small personal computers. This open source operating system inspired Linus Torvalds' initial development of Linux in the early 1990s (see Figure 1).

Figure 1. Short history of major Linux kernel releases

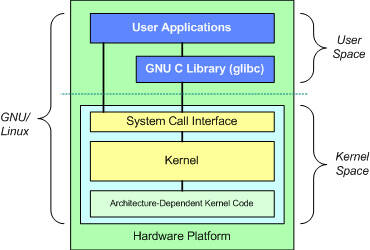


Linux quickly evolved from a single-person project to a world-wide development project involving thousands of developers. One of the most important decisions for Linux was its adoption of the GNU General Public License (GPL). Under the GPL, the Linux kernel was protected from commercial exploitation, and it also benefited from the user-space development of the GNU project (of Richard Stallman, whose source dwarfs that of the Linux kernel). This allowed useful applications such as the GNU Compiler Collection (GCC) and various shell support.

Introduction to the Linux kernel

Now on to a high-altitude look at the GNU/Linux operating system architecture. You can think about an operating system from two levels, as shown in Figure 2.

Figure 2. The fundamental architecture of the GNU/Linux operating system



**Methods for system call interface (SCI)**

In reality, the architecture is not as clean as what is shown in Figure 2. For example, the mechanism by which system calls are handled (transitioning from the user space to the kernel space) can differ by architecture. Newer x86 central processing units (CPUs) that provide support for virtualization instructions are more efficient in this process than older x86 processors that use the traditional int 80h method.

At the top is the user, or application, space. This is where the user applications are executed. Below the user space is the kernel space. Here, the Linux kernel exists.

There is also the GNU C Library (glibc). This provides the system call interface that connects to the kernel and provides the mechanism to transition between the user-space application and the kernel. This is important because the kernel and user application occupy different protected address spaces. And while each user-space process occupies its own virtual address space, the kernel occupies a single address space.

The Linux kernel can be further divided into three gross levels. At the top is the system call interface, which implements the basic functions such as read and write. Below the system call interface is the kernel code, which can be more accurately defined as the architecture-independent kernel code. This code is common to all of the processor architectures supported by Linux. Below this is the architecture-dependent code, which forms what is more commonly called a BSP (Board Support Package). This code serves as the processor and platform-specific code for the given architecture.

Properties of the Linux kernel

When discussing architecture of a large and complex system, you can view the system from many perspectives. One goal of an architectural decomposition is to provide a way to better understand the source, and that's what we'll do here.

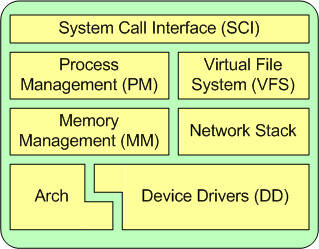
The Linux kernel implements a number of important architectural attributes. At a high level, and at lower levels, the kernel is layered into a number of distinct subsystems. Linux can also be considered monolithic because it lumps all of the basic services into the kernel. This differs from a microkernel architecture where the kernel provides basic services such as communication, I/O, and memory and process management, and more specific services are plugged in to the microkernel layer. Each has its own advantages, but I'll steer clear of that debate.

Over time, the Linux kernel has become efficient in terms of both memory and CPU usage, as well as extremely stable. But the most interesting aspect of Linux, given its size and complexity, is its portability. Linux can be compiled to run on a huge number of processors and platforms with different architectural constraints and needs. One example is the ability for Linux to run on a process with a memory management unit (MMU), as well as those that provide no MMU. The uClinux port of the Linux kernel provides for non-MMU support.

Major subsystems of the Linux kernel

Now let's look at some of the major components of the Linux kernel using the breakdown shown in Figure 3 as a guide.

Figure 3. One architectural perspective of the Linux kernel



System call interface

The SCI is a thin layer that provides the means to perform function calls from user space into the kernel. As discussed previously, this interface can be architecture dependent, even within the same processor family. The SCI is actually an interesting function-call multiplexing and demultiplexing service. You can find the SCI implementation in ./linux/kernel, as well as architecture-dependent portions in ./linux/arch.

Process management

**What is a kernel?**

As shown in [Figure 3](https://developer.ibm.com/articles/l-linux-kernel/#figure3), a kernel is really nothing more than a resource manager. Whether the resource being managed is a process, memory, or hardware device, the kernel manages and arbitrates access to the resource between multiple competing users (both in the kernel and in user space).

Process management is focused on the execution of processes. In the kernel, these are called *threads* and represent an individual virtualization of the processor (thread code, data, stack, and CPU registers). In user space, the term *process* is typically used, though the Linux implementation does not separate the two concepts (processes and threads). The kernel provides an application program interface (API) through the SCI to create a new process (fork, exec, or Portable Operating System Interface [POSIX] functions), stop a process (kill, exit), and communicate and synchronize between them (signal, or POSIX mechanisms).

Also in process management is the need to share the CPU between the active threads. The kernel implements a novel scheduling algorithm that operates in constant time, regardless of the number of threads vying for the CPU. This is called the O(1) scheduler, denoting that the same amount of time is taken to schedule one thread as it is to schedule many. The O(1) scheduler also supports multiple processors (called Symmetric MultiProcessing, or SMP). You can find the process management sources in ./linux/kernel and architecture-dependent sources in ./linux/arch).

Memory management

Another important resource that's managed by the kernel is memory. For efficiency, given the way that the hardware manages virtual memory, memory is managed in what are called *pages* (4KB in size for most architectures). Linux includes the means to manage the available memory, as well as the hardware mechanisms for physical and virtual mappings.

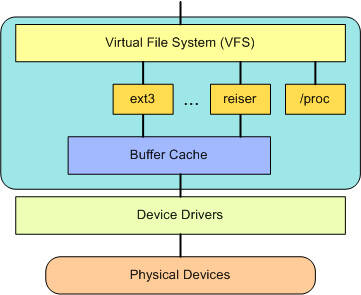
But memory management is much more than managing 4KB buffers. Linux provides abstractions over 4KB buffers, such as the slab allocator. This memory management scheme uses 4KB buffers as its base, but then allocates structures from within, keeping track of which pages are full, partially used, and empty. This allows the scheme to dynamically grow and shrink based on the needs of the greater system.

Supporting multiple users of memory, there are times when the available memory can be exhausted. For this reason, pages can be moved out of memory and onto the disk. This process is called *swapping* because the pages are swapped from memory onto the hard disk. You can find the memory management sources in ./linux/mm.

Virtual file system

The virtual file system (VFS) is an interesting aspect of the Linux kernel because it provides a common interface abstraction for file systems. The VFS provides a switching layer between the SCI and the file systems supported by the kernel (see Figure 4).

Figure 4. The VFS provides a switching fabric between users and file systems



At the top of the VFS is a common API abstraction of functions such as open, close, read, and write. At the bottom of the VFS are the file system abstractions that define how the upper-layer functions are implemented. These are plug-ins for the given file system (of which over 50 exist). You can find the file system sources in ./linux/fs.

Below the file system layer is the buffer cache, which provides a common set of functions to the file system layer (independent of any particular file system). This caching layer optimizes access to the physical devices by keeping data around for a short time (or speculatively read ahead so that the data is available when needed). Below the buffer cache are the device drivers, which implement the interface for the particular physical device.

Network stack

The network stack, by design, follows a layered architecture modeled after the protocols themselves. Recall that the Internet Protocol (IP) is the core network layer protocol that sits below the transport protocol (most commonly the Transmission Control Protocol, or TCP). Above TCP is the sockets layer, which is invoked through the SCI.

The sockets layer is the standard API to the networking subsystem and provides a user interface to a variety of networking protocols. From raw frame access to IP protocol data units (PDUs) and up to TCP and the User Datagram Protocol (UDP), the sockets layer provides a standardized way to manage connections and move data between endpoints. You can find the networking sources in the kernel at ./linux/net.

Device drivers

The vast majority of the source code in the Linux kernel exists in device drivers that make a particular hardware device usable. The Linux source tree provides a drivers subdirectory that is further divided by the various devices that are supported, such as Bluetooth, I2C, serial, and so on. You can find the device driver sources in ./linux/drivers.

Architecture-dependent code

While much of Linux is independent of the architecture on which it runs, there are elements that must consider the architecture for normal operation and for efficiency. The ./linux/arch subdirectory defines the architecture-dependent portion of the kernel source contained in a number of subdirectories that are specific to the architecture (collectively forming the BSP). For a typical desktop, the i386 directory is used. Each architecture subdirectory contains a number of other subdirectories that focus on a particular aspect of the kernel, such as boot, kernel, memory management, and others. You can find the architecture-dependent code in ./linux/arch.

Interesting features of the Linux kernel

If the portability and efficiency of the Linux kernel weren't enough, it provides some other features that could not be classified in the previous decomposition.

Linux, being a production operating system and open source, is a great test bed for new protocols and advancements of those protocols. Linux supports a large number of networking protocols, including the typical TCP/IP, and also extension for high-speed networking (greater than 1 Gigabit Ethernet [GbE] and 10 GbE). Linux also supports protocols such as the Stream Control Transmission Protocol (SCTP), which provides many advanced features above TCP (as a replacement transport level protocol).

Linux is also a dynamic kernel, supporting the addition and removal of software components on the fly. These are called dynamically loadable kernel modules, and they can be inserted at boot when they're needed (when a particular device is found requiring the module) or at any time by the user.

A recent advancement of Linux is its use as an operating system for other operating systems (called a hypervisor). Recently, a modification to the kernel was made called the Kernel-based Virtual Machine (KVM). This modification enabled a new interface to user space that allows other operating systems to run above the KVM-enabled kernel. In addition to running another instance of Linux, Microsoft® Windows® can also be virtualized. The only constraint is that the underlying processor must support the new virtualization instructions.

Going further

This article just scratched the surface of the Linux kernel architecture and its features and capabilities. You can check out the Documentation directory that's provided in every Linux distribution for detailed information about the contents of the kernel.

**linux architecture**

**By Neil Mitra**

13 mins read

Last updated: 4 May 2023

377 views

Topics Covered

**Overview**

Linux is a free, open-source and one of the most popular operating systems in the world, powering everything from mobile devices and desktops to servers and supercomputers. The Linux architecture is a modular architecture thatq acts as the backbone of the Linux operating system and allows its users to customize and configure the system to meet their specific needs. The Linux architecture diagram consists of several components including kernel, shell and applications, and is designed to be flexible, modular, and scalable, with support for a wide range of hardware platforms and applications.

**What is the Linux operating system?**

Linux is an open-source operating system that has been developed and maintained by a global community of developers for more than three decades. It was created in 1991 by Linus Torvalds, based on the Unix operating system and soon became known for its stability, security, and customizability. Linux is a kernel, which means that it manages the system's resources, such as the CPU, memory, and input/output operations.

The Linux operating system has a unique architecture that is designed to be modular, scalable and flexible, making it an ideal platform for a wide range of applications, from desktop computing to cloud infrastructure. Since It is open-source, anyone can contribute to its development, and its source code is available for anyone to view, modify and distribute. This makes the operating system popular among developers and IT professionals.

Linux can run on a wide range of hardware, from desktop computers to mobile devices. It offers a wide range of applications and utilities, allowing users to customize their experience to meet their specific needs. Linux is also known for its security, as it is inherently more secure than other operating systems due to its open-source nature, which allows for constant security updates and patches. Overall, Linux is a powerful and versatile operating system that is a favorite among users who value stability, security, and customizability. To learn more about the Linux operating system, you can visit [here](https://www.scaler.com/topics/linux-operating-system/).

**Features of Linux Operating System**

Let us go through some of the core features of Linux to build a better understanding about the operating system:

* **Open-Source:** Linux is free and open-source, meaning that the source code is available for anyone to view, modify, and distribute. This has led to a vast community of developers who contribute to the development and improvement of the operating system.
* **Flexibility:** Linux is highly customizable, allowing users to modify and tailor the operating system to meet the needs of specific applications. This flexibility makes Linux an ideal platform for a wide range of applications, from desktop computing to scientific research.
* **Scalability:** Linux can run on a wide range of hardware platforms, from small embedded devices to large-scale server clusters. This scalability makes Linux an ideal platform for cloud computing and other large-scale applications.
* **Security:** Linux is known for its robust security features, including support for secure boot, encryption, and access controls. These security features make Linux an ideal platform for sensitive applications, such as financial transactions and military applications.
* **Stability:** Linux is known for its stability and reliability, with many systems running for years without any downtime or issues. This stability makes Linux an ideal platform for critical applications, such as servers and medical devices.
* **File Systems:** Linux supports a variety of file systems, including Ext4, XFS, and Btrfs. These file systems offer robust features such as journaling, compression, and snapshotting, which can help improve performance, reliability, and data integrity.
* **Package Management:** Linux comes with a powerful package management system that makes it easy to install, upgrade, and remove software packages. This system provides a centralized repository of software that can be easily accessed and managed by users.

**Architecture of Linux system**

The architecture of the Linux system is a layered structure that comprises several components, each with a specific function. In simple terms, it can be coined as the way different parts of the Linux operating system work together to make a computer function.

The Linux architecture diagram provides a high-level overview of the various layers and components that make up the Linux operating system.

The various components of the Linux architecture are:

**Kernel**

The kernel is the central component of the Linux operating system architecture. It manages system resources and provides services to applications and processes running on the system. The Linux kernel is a monolithic kernel, which means that it contains all the core operating system functionality in a single executable file. It is highly modular and configurable, supporting a wide range of hardware devices and system configurations.

The kernel provides system calls, which allow applications and processes to interact with the operating system, including access to file management, process management, network communication, and hardware device control. The Linux kernel also includes support for third-party software such as device drivers, file systems, and network protocols, making it possible for Linux to run on a wide range of hardware platforms and support a diverse range of applications and use cases.

**Device Drivers**

Device drivers are an essential part of the Linux architecture, enabling the operating system to communicate with various hardware components and provide support for a wide range of devices. These act as an interface between the operating system and hardware devices, enabling communication between the two. The drivers translate commands and instructions from the operating system into a language that the device can understand.Linuxsupports two types of device drivers: character device drivers and block device drivers. Character device drivers enable data transfer in a stream, while block device drivers manage the transfer of data in fixed-size blocks.

Device drivers are built into the kernel or loaded as modules at runtime, providing support for a wide range of hardware components. Linux also supports open-source device drivers, allowing for easier development, modification, and customization. This open-source nature of the Linux operating system enables device driver developers to collaborate and share their work, resulting in a wide range of supported hardware components.

**System Libraries**

System libraries in the Linux architecture are pre-written code modules that can be used by applications to perform common functions. There are two types of libraries in Linux, static and dynamic, with dynamic libraries being preferred as they can be shared between multiple applications, reducing the memory footprint of the system. The most common system libraries in Linux include the C standard library, the GNU C Library, OpenSSL library for secure communication, GTK library for creating graphical user interfaces, and ALSA library for audio processing.

The use of system libraries in Linux allows for efficient and robust application development by leveraging pre-existing and optimized code. By providing a standard set of libraries, Linux ensures greater interoperability and consistency across the system. Overall, system libraries are an essential part of the Linux architecture, enabling developers to write efficient and optimized applications with ease.

**System Calls**

System calls are an important part of the Linux architecture that allow applications to interact with the kernel and manage system resources like memory, file systems, and devices. They are implemented as functions within the kernel that are called by applications through a standardized interface defined by the POSIX standard. The interface includes system call numbers that identify specific system calls. Common system calls in Linux include open(), read(), write(), close(), fork(), and exec().

System calls are essential for managing system resources, allowing applications to request access to resources and interact with them in a standardized way. They also promote compatibility and interoperability across different applications and hardware platforms. Overall, system calls play a critical role in the Linux architecture, enabling applications to manage system resources efficiently and reliably.

**Shell**

In Linux, the shell is a command-line interface that allows users to interact with the system through a set of commands. The most popular shell in Linux is the Bash shell, which communicates with the kernel to execute commands and manage system resources. The shell provides a set of built-in commands and the ability to execute external programs. Users can also customize the environment and automate tasks using shell scripts.

The shell is a powerful tool for managing system resources, and allows users to chain commands together to perform complex tasks. Overall, it is an essential component of the Linux architecture, providing a flexible and powerful command-line interface that allows users to interact with the system in a variety of ways, automate tasks, and manage system resources efficiently.

**Applications:**

Applications in the Linux architecture are software programs that run on top of the operating system and interact with its components. Linux offers a wide range of applications that cover almost every aspect of computing, including web browsers, office suites, media players, graphics editors, and text editors. These applications are developed using various programming languages and are designed to work efficiently and effectively on different hardware architectures, making them ideal for use on a wide range of devices. These applications also provide a diverse range of functionality for users, and the flexibility and customizability of Linux enable developers to create new applications and modify existing ones to meet their specific needs.

Overall, Linux applications offer a powerful and versatile set of tools that enable users to complete a wide range of tasks, from web browsing and office productivity to graphic design and media playback.

**Graphical User Interface**

The Graphical User Interface (GUI) in the Linux architecture is a desktop environment that provides users with a more intuitive and user-friendly way to access system resources and manage applications. It is built using the X Window System, which provides the underlying infrastructure for graphics rendering and display management. Many Linux distributions use a display manager to manage the login and session management process. However, the use of the GUI in Linux is optional, and many users prefer to work from the command line.

The GUI provides a flexible and customizable desktop environment while using an operating system. There are several desktop environments available in Linux, each with its own unique look and feel, including GNOME, KDE, Xfce, and LXDE. Overall, the GUI is an important component of the Linux architecture that provides a more accessible and user-friendly way to interact with the system, while also offering the flexibility to customize and modify the desktop environment to meet specific needs.

**Advantages of Linux**

Using the Linux operating system has many advantages. Some notable advantages of Linux are mentioned below -

* **Cost Effective:** One of the key advantages of Linux is its cost-effectiveness, as it is generally free and can be installed on any number of computers without incurring licensing fees. Linux's open-source nature also provides users with access to the source code, allowing for greater customization and cost-effective software solutions.
* **Security:** Linux is highly secure, thanks to its built-in firewall, file permissions system, and encrypted file system support. Its open-source nature allows for quick identification and resolution of security vulnerabilities, making it a trusted and secure platform for a range of applications. The collaborative approach to security, with the community of developers working together to address threats, ensures that Linux stays secure and reliable for servers, workstations, and embedded devices.
* **Stability:** Another advantage of Linux is its stability. Linux is known for its robust and stable performance, which can result in fewer system crashes and less downtime. This stability is due in part to the modular design of the Linux kernel, which allows developers to identify and fix bugs quickly.
* **Flexibility:** Flexibility is another advantage of Linux. The modular nature of Linux allows it to be tailored to the specific needs of different users and applications. The Linux kernel can be customized and configured to support different hardware and software configurations, making it ideal for use in a variety of devices and systems.
* **Performance:** Flexibility is another advantage of Linux. The modular nature of Linux allows it to be tailored to the specific needs of different users and applications. The Linux kernel can be customized and configured to support different hardware and software configurations, making it ideal for use in a variety of devices and systems.

**Drawbacks of Linux**

Although the advantages of Linux often outweigh its issues, it does have a few drawbacks. Some of the drawbacks of the Linux OS are mentioned below -

* **User interface:** The user interface can be mentioned as a drawback of Linux, as it can be less user-friendly than other operating systems, especially for those unfamiliar with the command-line interface or Linux desktop environment. However, user-friendly Linux distributions such as Ubuntu and Linux Mint offer a more familiar interface. Additionally, Linux provides a high degree of customization, allowing users to tailor the interface to their preferences with themes and other options.
* **Application availability:** The limited availability of commercial applications and games is a drawback of Linux. While many open-source alternatives are available, certain proprietary software may not be optimized for Linux or have limited functionality. However, the availability of applications and games on Linux is improving, and compatibility tools like Wine and Proton can be used to run Windows applications and games on Linux with varying success.
* **Hardware support:** Hardware support can be a drawback of Linux, since some hardware devices are not fully supported, and some may require additional configuration or custom drivers, which can delay or prevent Linux users from upgrading their systems. However, Linux offers broad hardware support, and many manufacturers are now offering Linux drivers and support.
* **Learning curve:** Another drawback of Linux is the learning curve, as it can be more challenging for users who are not familiar with the command-line interface or the Linux desktop environment. Linux may require users to learn new commands and processes, which can be time-consuming and at certain times frustrating.

**Linux Operating System Applications**

Linux provides a vast variety of applications for its users. Some of the most popular applications available on Linux are given below:

* **LibreOffice:** LibreOffice is a free and open-source office suite that includes a word processor, spreadsheet program, presentation software, and more. It provides many of the features of popular commercial office suites and is compatible with a wide range of file formats.
* **GIMP:** GIMP (GNU Image Manipulation Program) is a free and open-source image editing software that provides advanced features and tools for image manipulation. It can be used for tasks like photo retouching, image composition, and graphic design.
* **VLC Media Player:** VLC Media Player is a free and open-source media player that can play almost any media format. It provides a simple and intuitive interface and supports advanced features like subtitles and streaming.
* **Thunderbird:** Thunderbird is an open-source email client developed by Mozilla that allows users to send, receive and manage email messages. It is a feature-rich application that offers a wide range of functionalities such as email filtering, tagging, and virtual folders.
* **Vim:** Vim is a popular and powerful text editor used in Linux and other operating systems. It is a command-line interface-based editor that offers advanced features such as syntax highlighting, macros, plugins, and more, making it a preferred choice for developers and system administrators. It is highly configurable and efficient, with extensive documentation and community support.
* **Blender:** Blender is a free and open-source 3D modeling and animation software used to create 3D graphics, animation, and visual effects. It is a powerful tool for industries such as film, game development, and architecture. It offers advanced features like a flexible interface, modeling tools, a built-in game engine, and supports different rendering engines.

**Conclusion**

* Linux is a powerful and versatile open-source operating system that offers scalability, modularity, and security, making it an ideal platform for a wide range of applications.
* The Linux architecture is highly modular and configurable, offering a high degree of customization and flexibility to users and developers. It is a layered structure comprising several components, including the kernel, system libraries, system calls, shell, applications, graphical user interface, and device drivers.
* Advantages of Linux include cost-effectiveness, high security, stability, flexibility, and performance due to its open-source nature, modular design, and ability to be customized for specific needs.
* Linux has some drawbacks, such as, having a less user-friendly user interface for users who are unfamiliar with command-line interface or Linux desktop environment, limited availability of commercial applications and games, hardware support issues, and a challenging learning curve for new users.
* Linux offers a vast array of applications for various purposes, including office suites, web browsers, media players, and more. These applications are often free, open-source, and highly customizable, making Linux a popular choice for users seeking flexibility and cost-effectiveness.

**MCQs**

1. **Which one is an advantage of the Linux operating system?**

* a. Cost Effectiveness
* b. Security
* c. Stability
* d. All of the above
* e. Correct answer: <d. all of the above>

1. **Which component of the Linux architecture acts as an interface between the operating system and hardware devices?**

* a. System Libraries
* b. Shell
* c. Device Drivers
* d. Graphical User Interface
* e. Correct answer: <c. Device Drivers>

1. **Which type of kernel is used in Linux architecture?**

* a. Microkernel
* b. Monolithic Kernel
* c. Hybrid Kernel
* d. Exokernel
* e. Correct answer: <b. Monolithic Kernel>

Linux is an Operating System Kernal that is the core of many distributions and Operating systems. Its development started in the early 1990s and has been developing regularly since then. It is a free and Open Source software. Although it has relatively less market share than other major Operating Systems like Windows, it is the most used Operating System in the development world.

In this article, we will first go through the overview of Linux, its features, and then the detailed architecture of the system.

**What is Linux**

Linux is an open-source kernel developed by **Linus Torvalds**for efficiently managing various operations in computers. A kernel is a piece of software that allows computer hardware and software to communicate with one another. It transports input to the CPU for processing and output to the hardware for display. This is the most fundamental function of a Linux and operating system combined.

This OS communicates with users through ***Application Programming Interface (API)***. APIs are assembly-level codes that act as the intermediate between users and Kernel. APIs provide an abstraction of large and complex low-level instructions for the users.

[**Features of Linux**](https://www.techgeekbuzz.com/blog/features-of-linux/)

**1. Multitasking**

Multiple processes can be run concurrently by scheduling them and not leaving the CPU idle for long periods of time.

**2. User Management**

Multiple users can work together on a single Operating System without interfering with others. They cannot read or write other users’ data without permission. This feature is very helpful for companies and organizations since multiple people may work on a single OS.

**3. Multiple Desktop Environments**

We have a wide variety of Desktop Environments available in Linux Operating Systems. These include ***Gnome***, ***Mate***, ***Cinnamon***, ***KDE Plasma***, etc.

**4. Open Source**

It is open-source software that can be configured by everyone. This ensures quick bug removals and the introduction of innovative features in the tool.

**5. Hierarchical File System**

The filesystem is distributed in the form of a [tree data structure](https://www.techgeekbuzz.com/tutorial/data-structure/tree-data-structure-algorithm/)which ensures fast search, deletion, and insertion of files.

**6. Multithreading**

A thread of execution is the shortest succession of programmed instructions, each with its own set of program counters and registers. Threads enable many tasks to be completed in the same program. For example, we can use Google Chrome to both download and surf the web.

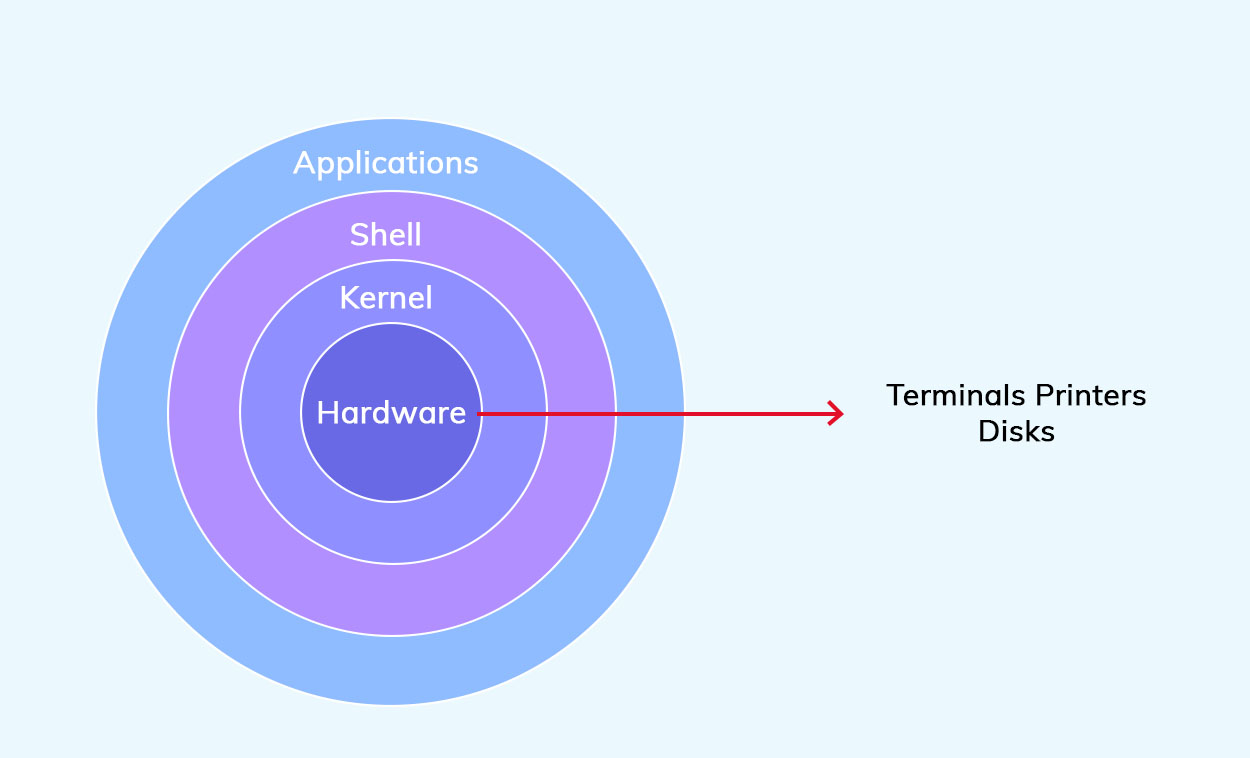
**Why Study Linux Architecture**

Understanding the architecture of an operating system ensures that work is done in an efficient and secure manner. You are aware of anything that may provide a security risk to your computer, as well as what is compatible with your existing architecture. This also saves a significant amount of online surfing time.

A solid grasp of the architecture also allows you to configure the system based on your use case and even contribute to the source code. As a result, it is usually suggested that you are familiar with the architecture of the system on which you are working.

**Architecture of Linux**

The below diagram shows how architecture is distributed in Linux.



User programs operate in a non-privileged manner. They must make a system call to the kernel for each job. This system call is made indirectly, using a library or an Application Programming Interface (API).

The API includes assembly code that serves as an abstraction for users to interface with hardware. The API instruction is subsequently sent to the Kernel, which has full access to physical devices. The mode changes from user mode to kernel mode at this time. A system call is then implemented by the kernel. The kernel is in charge of scheduling processes based on their priority.

Let's take a closer look at the components of the Linux architecture.

**1. Utilities/Applications**

These are the applications and software that are running on the screen. These can be Browsers, Video Games, etc. These do not have direct access to the hardware for security concerns and requests to access hardware devices through system calls.

**2. Shell**

Shell is a program that is used in UNIX-like systems to implement various instructions. It gives a computer user access to the Unix/GNU Linux system, allowing the user to perform various commands or utilities/tools with some input data. When the shell finishes executing [a program](https://www.techgeekbuzz.com/blog/what-is-a-program/), it displays the result to the user on the standard output device, the screen.

As a result, it is known as the "command interpreter." The default shell in Linux is "Bash," which stands for "Bourne Again Shell" and is a successor for Bourne Shell.

**3. Application Programming Interface (API) or Libraries**

An Application Program Interface (API) is a set of instructions that enables software applications to communicate with one another. It specifies how to create a program that seeks services from an operating system or another application. Function calls are used to implement APIs.

In general, it is a set of explicitly specified communication mechanisms between diverse software components. It encapsulates the underlying functionality and only exposes the objects or activities required by the developer, minimizing the workload on programmers. When you make a system call, it is passed through the APIs defined in every Operating System.

**What is a System Call in Linux architecture?**

System calls allow user applications (running in user mode) to request a service from the operating system. In other words, system calls enable user applications to request that the operating system perform certain tasks on their behalf. For example, read and write file operations that necessitate I/O from/to the storage device. Such system call actions are available to the end-user through basic library calls/APIs such as ***read()***, ***write()***, ***open()***, and so on.

The OS kernel code executes the meat of such actions after switching the mode ( **TRAP instruction**) from user to kernel. These and other system call procedures often access/manipulate OS kernel data structures such as the buffer cache, process control block, global open file table, and many more. Such kernel-level data structures are crucial to the system's overall functionality. This is why programs running in the kernel should be allowed to access them instead of users themselves.

**4. Kernel**

A kernel is a software that controls the hardware and offers services to other programs in order to mediate all access to the hardware (including memory allocation). As a result, the kernel serves as the hub of all activity for all applications executing on that operating system.

Kernels and microkernel designs are typical for multi-user and multi-tasking systems and may therefore enforce resource constraints to safeguard the OS and other applications from any program (or user) that is either defective or malicious. The Linux kernel is monolithic, which implies that all of the kernel's services execute concurrently with the main thread. It is easier to implement a monolithic kernel but it suffers major issues if even a single device driver fails.

**a. Monolithic Kernel**

It handles all of the core system services, including processes and their scheduling, interrupts handling and access to the hardware, file system, and so on itself. It is designed in layers, beginning with basic process management and progressing to interfaces with the rest of the operating system including the libraries and the applications.

**b. Microkernel**

This kernel employs the bare minimum of configuration for scheduling, memory management, and inter-process communication. This significantly decreases the amount of memory needed for kernel use. Because the kernel is as small as possible, the amount of communication that is needed by the device drivers limits the flow of data via the kernel while simultaneously decreasing kernel reaction time to interruptions. Microkernels are common in ***real-time systems***.

**c. Hybrid Kernel**

The kernel is bigger than a microkernel but smaller than a monolithic kernel. Typically, you'll receive a stripped-down monolithic kernel with the bulk of device drivers eliminated but all of the system services remaining there in the kernel space. These kernels are commonly found on desktop computers running ***Windows***, ***Mac***, or ***Linux***. Below is the image from Wikipedia showing the distinction between the monolithic kernel, micro, and hybrid kernel.

**5. Hardware**

This comprises all physical equipment linked to the computers, such as hard discs, RAM, CPUs, and so on. The kernel uses these devices for a variety of hardware-related operations like reading and writing files, CPU processing, creating and stopping processes. This is the final stage of the Linux instruction flow, where the operation is carried out.

**6. Bootloader**

A bootloader is a piece of code/program that runs prior to the start of an operating system. When a computer is turned on, it loads an operating system. When you start things up, it instructs the hardware where to look and how to begin going. The bootloader essentially packages the instructions to boot the operating system kernel, and most of them also include their own debugging or modification environment.

**Conclusion**

This article goes over the details of Linux architecture. Because of its essential characteristics such as security, identity and access control, configuration, and variety, Linux is a kernel that is extensively used for development today. Linux's architecture consists of hardware, the Kernel, APIs, the Shell, and applications.

The Kernel is the fundamental component of an operating system that implements user-requested system calls. The APIs act as middlemen between user and kernel level modes. The hardware is the end device with which the Kernel communicates in order to perform different tasks.