What is the Linux Kernel?

Introduction to Linux Kernel

The **Linux kernel** is an open-source, free, multitasking, modular, monolithic, and Unix-like OS kernel. In 1991, it was originally started by **Linus Torvalds** for his i386-based PC and was soon adopted for the GNU operating system as the kernel, which was written to be a libre (free) replacement for Unix.

Linux is given under the GNU General Public License v2 only, but it includes files in other compatible licenses. It has been added as a part of several operating system distributions since late 1990, several of which are also known as Linux commonly.

- ❍Linux is used on a huge range of computing systems, like supercomputers, mainframes, servers, personal computers, mobile devices, and embedded devices.
- ❍It can be tailored for many usage scenarios and architectures with a family of easy commands; privileged users can fine-tune kernel parameters during runtime as well.
- ❍Almost every code of the Linux kernel is written with the GCC GNU extensions to the C programming language.
- ❍It generates a highly **vmlinux** (optimized executable) to utilize the task execution time and memory space.
- ❍Day-to-day development discussions consider on the **LKML** (Linux Kernel mailing list).
- ❍Modifications are tracked with the git version control system, which was originally started by Torvalds for **BitKeeper** as a free software replacement.

Features and Architecture of Linux Kernel

Linux is a monolithic kernel with a standard design, supporting almost every feature once only present in closed source non-free operating system kernels. The following subsequent sections and list specify a non-comprehensive introduction to Linux architectural design and a few of its important aspects:

- ❍Concurrent computing and true parallel execution of several processes at once on NUMA and SMP architectures.

❍Configuration and selection of several kernel drivers and features, kernel parameter modification before booting, and kernel behavior fine-tuning during runtime.

❍Configuration and runtime changes of the policies of the task schedulers that permit preemptive multitasking; CFS (Completely Fair Scheduler) has been the default scheduler for Linux since 2007, and it utilizes a red-black tree that can find, insert, and remove process information using O(log n) time complexity, in which n is the runnable tasks.

❍Synchronization mechanism and inter-process communications.

❍Advanced memory management using paged virtual memory.

❍A virtual filesystem on the head of many concrete file systems (FAT32, JFS, XFS, Btrfs, ext4, and several others).

❍Configurable input & output schedulers, ioctl(2) syscall that can manipulate basic device parameters of unique files, supports asynchronous input & output.

❍OS-level virtualization, hardware-assisted virtualization, and paravirtualization. The Linux Kernel supports to make Linux distributions that operate as Dom0 on the Xen hypervisor.

❍I/O Virtualization with SR-IOV and VFIO. VFIO (Virtual Function I/O) reveals direct device access in a secure memory-protected environment. A VM Guest can access hardware devices directly on the VM Host Server with VFIO. If compared Paravirtualization and Full virtualization, this method improves performance.

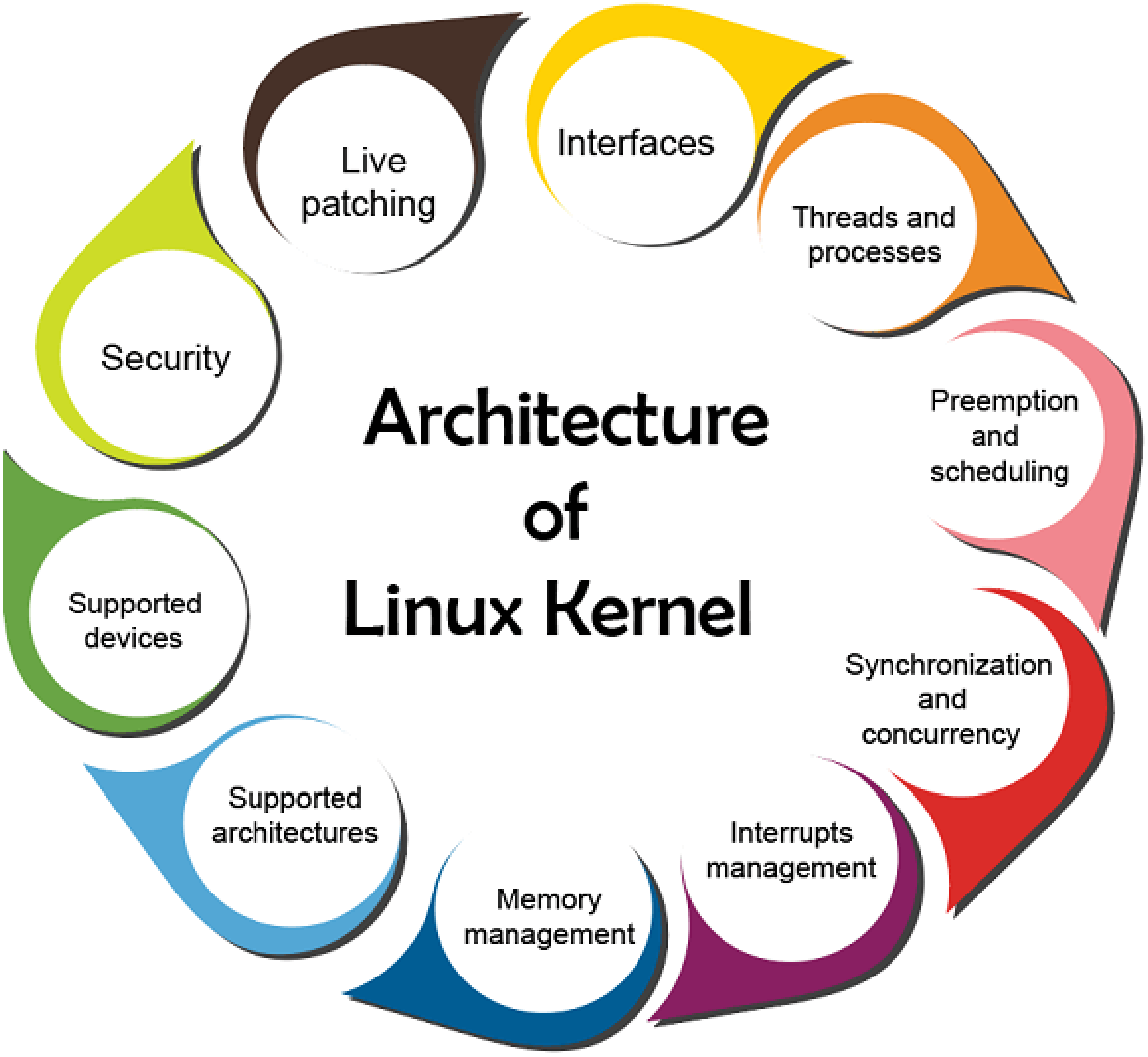
❍Security mechanisms for mandatory and discretionary access control (POSIX ACLs, AppArmor, SELinux, and others).

❍Asymmetric multiprocessing by the RPMsg subsystem.

❍Various layered communication protocols (such as the Internet protocol suite).

Most kernel extensions and device drivers active in kernel space with complete access to the hardware. A few exceptions active in user space. Furthermore, the **Wayland** and **X Window System**, the display server protocols and windowing system that almost all people utilize with Linux.

Device drivers are configured as modules and unloaded or loaded while the system is active and can also be preempted in certain conditions to correctly manage hardware interrupts and to better support symmetric multiprocessing, unlike standard monolithic kernels. Linux uses virtual memory and memory protection and can also manage non-uniform memory access.



Interfaces

Linux is a copy of Unix and focuses on Single UNIX and POSIX specification compliance. Also, the kernel offers Linux-specific system calls and several interfaces. The code must satisfy a group of licensing rules to be added to the official kernel.

Linux ABI (Application Binary Interface) between the user space and the kernel has four stability degrees (removed, obsolete, testing, stable). Although, the system calls are supposed to never modify to not crack the userspace programs that depend on them.

LKMs (Loadable kernel modules) can't depend on a stable ABI by design. Hence, they should always be re-arranged whenever a fresh kernel executable is in a system. In-tree drivers, configured to be an integral segment of the vmlinux (kernel executable), are statically connected by the building process.

Threads and processes

Linux establishes processes using the newer clone3 (2) and clone (2) system calls. The new entity can share none or most of the resources of the caller, relying on the given parameters. These syscalls can establish new entities ranging from the new independent processes to new execution threads in the calling process.

If the executable is connected to shared libraries dynamically, a dynamic linker finds and loads the required objects, arranges the program to execute, and then executes it. A very unique thread category is the kernel threads. These must not be confused with the execution thread of the user's processes which are mentioned above. Kernel threads are available in kernel space only, and their purpose is to run kernel operations concurrently.

Preemption and scheduling

The Linux scheduler is standard as it enables distinct scheduling policies and classes. Scheduler classes can be defined as pluggable scheduler algorithms that can be enrolled with the basic scheduler code. All classes schedule several processes. The core scheduler code iterates over all classes in priority order and selects the highest priority scheduler that contains a schedulable struct entity type, i.e., **sched_entity**, ready to execute.

Entities may be groups of threads, threads, and even every process of a particular user. Linux offers both full kernel preemption and user preemption. Preemption decreases latency, enhances responsiveness, and enables Linux to be more compatible with real-time and desktop applications.

Synchronization and concurrency

The kernel has different concurrency causes (e.g., symmetrical multiprocessing, kernel preemption and user tasks, bottom halves, and interrupts). To protect critical regions and memory regions that are changeable by hardware asynchronously, Linux offers a large group of tools.

They are composed of atomic types, lockless, mutexes, semaphores, and spinlocks algorithms. Almost every lockless algorithm is established on memory barriers to enforce memory ordering and avoid unwanted side effects because of compiler optimization.

Interrupts management

Interrupt management is categorized into two different parts. However, it could be considered a single job. It splits into two due to the distinct time constraints, and synchronization requires tasks whose the management is consisted of. The initial part is composed of an interrupt service routine that is called the **top half** in Linux, while the second part is imposed by the **bottom halves**. The service routines of Linux interrupts can be nested.

Memory management

In Linux, memory management is a complex topic. The kernel isn't pageable. Memory protection is not available in the kernel; hence, memory violations cause system crashes and instability. Linux operates virtual memory with 4 and 5-level page tables. User memory space is pageable only. It manages information about all page frames of RAM in related data structures that are immediately populated after restarts and kept until shut down, despite being or not related to virtual pages.

Supported architectures

Now, Linux is one of the most highly ported OS kernels, running on different systems from an ARM Architecture to IBM z/Architecture mainframe computers while not actually developed to be portable. The initial port was implemented on the Motorola 68000 platform.

The changes to the kernel were so crucial that Torvalds saw the Motorola edition as a **"Linux-like operating system"** and a fork. However, that lifted Torvalds for leading a major code restructure to provide porting to several computing architectures. Also, Linux has been ported to several handheld devices like Apple's iPod and iPhone 3G.

Supported devices

The LKDDb project began to create a comprehensive database of protocols and hardware known by Linux kernels in 2007. The database is automatically built by the kernel source's static analysis. The Linux hardware project was announced to assemble a database of every tested hardware configuration automatically from the users of many Linux distributions later in 2014.

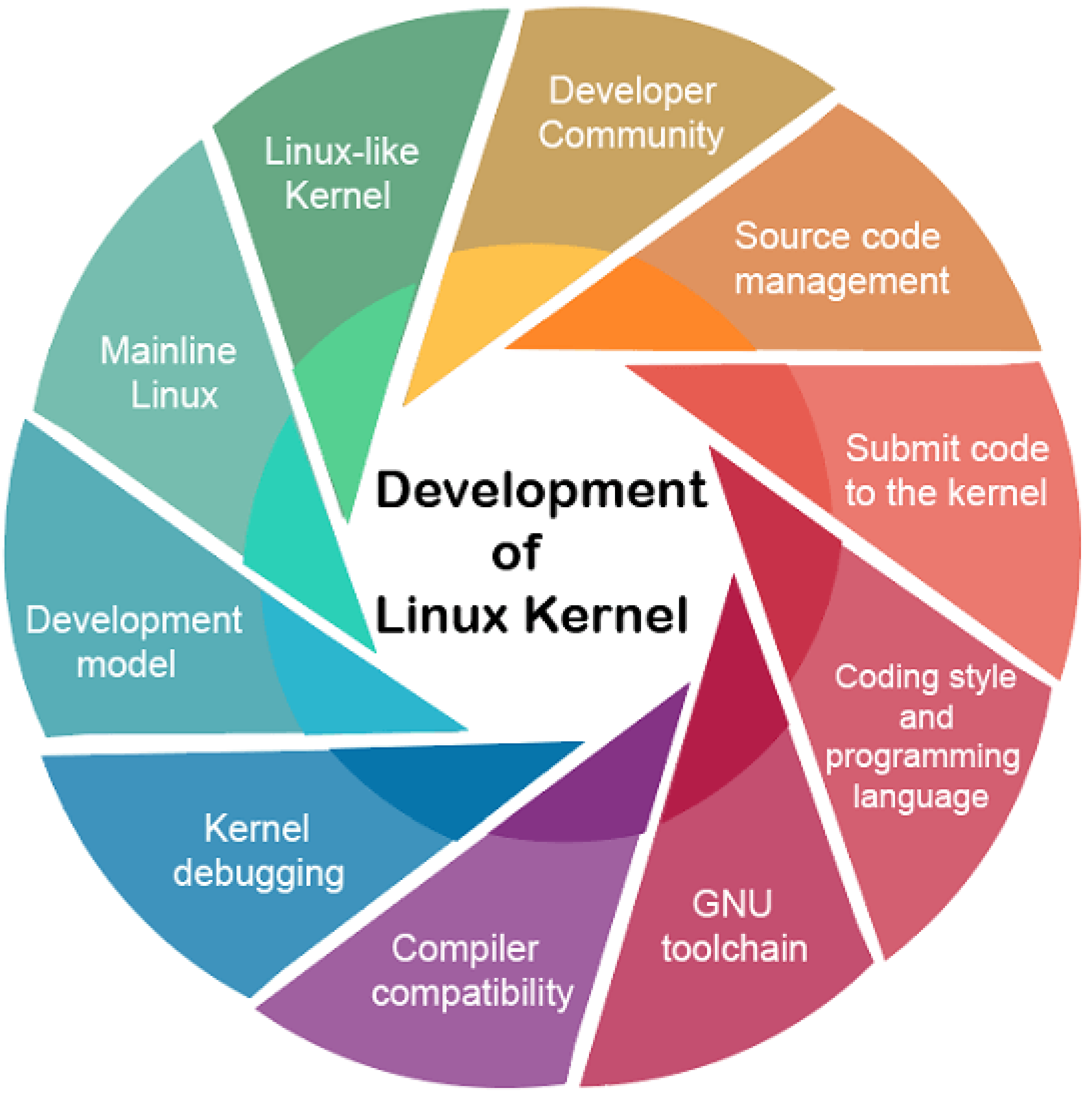
Live patching

In live patching, rebootless updates can be even used to the kernel with live patching methods like kGraft, kpatch, and ksplice. For live kernel patching, minimalistic foundations were combined into the mainline of Linux kernel in the 4.0 version of the kernel, which was published on 12 April 2015.

Security

In kernel, bugs present security problems. For instance, they may permit privilege escalation or make denial-of-service attack vectors. Several bugs are affecting system security that was detected and fixed over the years. New aspects are implemented to increase the security of the kernel frequently.

Development of Linux Kernel



Developer community

The developer community of Linux kernel is composed of about 5000 to 6000 members. The top 30 developers shared a little more than 16% of the code. Developers are needed to obey the Contributor Covenant as with various open-source application projects. Contributor Covenant is a code of conduct to specify the molestation of minority contributors. To prevent inclusive terminology, use the source code that is authorized.

Source code management

The community of Linux development utilizes Git to handle the source code. The users of Git copy the current release of Torvalds' tree with the help of git-clone(1) and keep it upgraded with git-pull(1).

Submit code to the kernel

The developer who wants to modify the Linux kernel begins with testing and developing that modification. Depending on how important the modification is and how many subsystems it changes, it will either be submitted as one patch or two or more source code patches.

In the case of one subsystem managed by one maintainer, the patches are transferred as emails of the subsystem to the maintainer with the correct mailing list in Cc. The reader and the maintainer of the mailing list will check the patches and give feedback. Once this review process has been completed, the subsystem maintainer adopts the patches in the related Git kernel tree.

Coding style and programming language

Linux OS is written in unique C language. The language is supported by GCC, which is a compiler that can extend in several ways the C standard. Every code must obey the 21 rules composed of the **Linux Kernel Coding Style** since 2002.

GNU toolchain

GNU cc or GCC (GNU Compiler Collection) is the default compiler for Linux sources, and it's required a utility known as make. The GNU Assembler results in the object files through the GCC produced assembly code. The GNU linker generates a statically connected executable kernel file known as vmlinux. Both ld and as are part of the GNU Binary Utilities. The tools are called GNU tools collectively which are mentioned above.

Compiler compatibility

For a long time, GCC was the only compiler able to correctly build Linux. Intel was likely to have changed the kernel so that its C compiler was able to compile it in 2004. Since 2010, the effort has been ongoing to establish Linux with Clang, a substitute compiler for the C language. After the LLVM compiler infrastructure under which Clang is established, the project embedded in this effort is called LLVMLinux.

Kernel debugging

Viruses affecting the Linux kernel can be hard to troubleshoot. It is due to the interaction of the kernel with hardware and userspace and also due to the fact they might be led from a huge variety of reasons than those of user programs. Some examples of the underlying reasons are improper hardware management, synchronization primitives misuse, and semantic errors in the code. In the kernel, a non-fetal bug report is known as an **"oops"**; such divergence from correct Linux kernel behavior may permit steady operation with agreed reliability.

Development model

Linux kernel accommodates new code on a rolling basis. Software inspected into the project must compile and work without error. All kernel subsystems are allotted a maintainer who is liable to review patches against the standards of the kernel code and keeps a patch queue that can be shared with Linus Torvalds in the merge window of many weeks. These patches are combined by Torvalds with the source code of the preceding stable release of the Linux kernel.

Mainline Linux

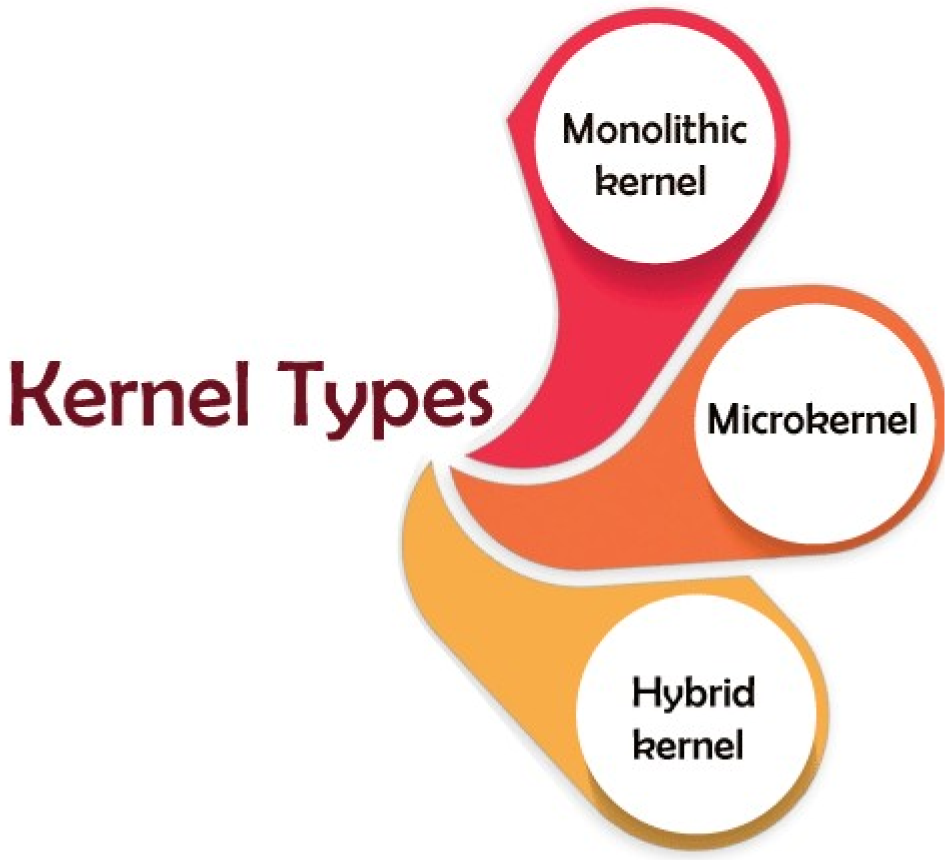
Linus Torvalds' Git tree is called mainline Linux, which includes the Linux kernel. All stable kernel releases come from the mainline tree, which is released on kernel.org frequently. Mainline Linux supports a small subset of several devices that execute Linux OS. Non-mainline support is given by independent projects like **Linaro** and **Yocto**, but in several cases, the kernel is required from the device vendor. The vendor kernel needs a board support package likely. Managing a kernel tree has proven to be hard external to the mainline Linux.

Linux-like Kernel

Greg Kroah-Hartman, the stable branch maintainer, has used the Linux-like term to downstream kernel splits by vendors that include several lines of code into the mainline kernel. Google said that they wished to utilize the mainline Linux kernel in the Android OS so the kernel forks would be decreased in 2019. Also, the Linux-like term has been used for the **Embeddable Linux Kernel Subset** that doesn't add the complete mainline Linux kernel but a tiny changed code subset.

Kernel Types

The kernel has three types in general, which are listed and explained below:



Monolithic kernel

It's a widely used kernel by OSes. The kernel is composed of several modules that can be loaded and unloaded dynamically in a monolithic architecture. This type of architecture would increase the capabilities of the operating system and permits easy extensions for the kernel. Kernel maintenance becomes convenient with monolithic architecture as it permits a concerned module for loading and unloading if there is a requirement to resolve a bug in a specific module.

Microkernel

Microkernel has been derived as a replacement for the monolithic kernel to define the problem of the ever-growing kernel code size, which the monolithic kernel was unsuccessful in doing. It permits a few basic services, such as file system, device driver management, protocol stack, etc., to execute in userspace. It could increase the capability of the operating system with improved security and minimum code and ensures stability.

It decreases the damage to the affected areas by making the rest of the system work without interruptions correctly. Every basic service of OS is there to programs by IPC (interprocess communication). In microkernel architecture. It permits direct interaction between the hardware and device drivers.

Hybrid kernel

It can determine what it wishes to execute in supervisor mode and user mode. Like device drivers, in the hybrid kernel environment, file system I/O would execute in user mode, although IPC and server calls reside in supervisor mode.

Userspace and kernel space

The system memory is categorized into two regions in the Linux OS: userspace and kernel space. Let's describe all regions of memory and understand the functionalities:

Userspace

The userland or userspace is a code that executes external to the operating system kernel platform. It is defined as several programs, applications, or libraries that an OS utilizes to link with the kernel. Due to the complicated process of using memory, mischievous functions can only be restricted to the user system.

Kernel space

It is found in a formal state, which offers complete access to the hardware equipment and keeps the memory space protected. This userspace and memory space together are known as kernel space. Core access to the system hardware and services is maintained and offered as a service for the rest of the system in the kernel space environment.