**Difference Between Linux And Unix:**

**Linux** is a **UNIX** Clone. But if you consider Portable Operating System Interface (POSIX) standards then **Linux** can be considered as **UNIX**. ... **Linux** is a **Unix** clone written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Net. It aims towards POSIX compliance.

## **Linux Is Just a Kernel**

Linux is just a kernel. All Linux distributions includes GUI system + GNU utilities (such as cp, mv, ls,date, bash etc) + installation & management tools + GNU c/c++ Compilers + Editors (vi) + and various applications (such as OpenOffice, Firefox). However, most UNIX operating systems are considered as a complete operating system as everything come from a single source or vendor.

As I said earlier Linux is just a kernel and Linux distribution makes it complete usable operating systems by adding various applications. Most UNIX operating systems comes with A-Z programs such as editor, compilers etc. For example HP-UX or Solaris comes with A-Z programs.

### Linux Distribution (Operating System) Names

A few popular names:

1. Redhat Enterprise Linux
2. Fedora Linux
3. Debian Linux
4. Suse Enterprise Linux
5. Ubuntu Linux

### UNIX Operating System Names

A few popular names:

1. HP-UX
2. IBM AIX
3. Sun Solairs
4. Mac OS X
5. IRIX

## Comparison chart

Linux versus Unix comparison chart

|  | Linux | Unix |
| --- | --- | --- |
|  |  |  |
| Cost | Linux can be freely distributed, downloaded freely, distributed through magazines, Books etc. There are priced versions for Linux also, but they are normally cheaper than Windows. | Different flavors of Unix have different cost structures according to vendors |
| Development and Distribution | Linux is developed by Open Source development i.e. through sharing and collaboration of code and features through forums etc and it is distributed by various vendors. | Unix systems are divided into various other flavors, mostly developed by AT&T as well as various commercial vendors and non-profit organizations. |
| Manufacturer | Linux kernel is developed by the community. Linus Torvalds oversees things. | Three bigest distributions are Solaris (Oracle), AIX (IBM) & HP-UX Hewlett Packard. And Apple Makes OSX, an unix based os.. |
| User | Everyone. From home users to developers and computer enthusiasts alike. | Unix operating systems were developed mainly for mainframes, servers and workstations except OSX, Which is designed for everyone. The Unix environment and the client-server program model were essential elements in the development of the Internet |
| Usage | Linux can be installed on a wide variety of computer hardware, ranging from mobile phones, tablet computers and [video game consoles](https://www.diffen.com/difference/PS4_vs_Wii_U), to mainframes and supercomputers. | The UNIX operating system is used in internet servers, workstations & PCs. Backbone of the majority of finance infastructure and many 24x365 high availability solutions. |
| File system support | Ext2, Ext3, Ext4, Jfs, ReiserFS, Xfs, Btrfs, FAT, [FAT32, NTFS](https://www.diffen.com/difference/FAT32_vs_NTFS) | jfs, gpfs, hfs, hfs+, ufs, xfs, zfs format |
| Text mode interface | BASH (Bourne Again SHell) is the Linux default shell. It can support multiple command interpreters. | Originally the Bourne Shell. Now it's compatible with many others including BASH, Korn & C. |
| What is it? | Linux is an example of Open Source software development and Free Operating System (OS). | Unix is an operating system that is very popular in universities, companies, big enterprises etc. |
| GUI | Linux typically provides two GUIs, [KDE and Gnome](https://www.diffen.com/difference/GNOME_vs_KDE). But there are millions of alternatives such as LXDE, Xfce, Unity, Mate, twm, ect. | Initially Unix was a command based OS, but later a GUI was created called Common Desktop Environment. Most distributions now ship with Gnome. |
| Price | Free but support is available for a price. | Some free for development use (Solaris) but support is available for a price. |
| Security | Linux has had about 60-100 viruses listed till date. None of them actively spreading nowadays. | A rough estimate of UNIX viruses is between 85 -120 viruses reported till date. |
| Threat detection and solution | In case of Linux, threat detection and solution is very fast, as Linux is mainly community driven and whenever any Linux user posts any kind of threat, several developers start working on it from different parts of the world | Because of the proprietary nature of the original Unix, users have to wait for a while, to get the proper bug fixing patch. But these are not as common. |
| Processors | Dozens of different kinds. | x86/x64, Sparc, Power, Itanium, PA-RISC, PowerPC and many others. |
| Examples | [Ubuntu](https://www.diffen.com/difference/Linux_Mint_vs_Ubuntu), [Fedora](https://www.diffen.com/difference/Fedora_vs_Ubuntu), Red Hat, Debian, Archlinux, Android etc. | OS X, Solaris, All Linux |
| Architectures | Originally developed for Intel's x86 hardware, ports available for over two dozen CPU types including ARM | is available on PA-RISC and Itanium machines. Solaris also available for x86/x64 based systems.OSX is PowerPC(10.0-10.5)/x86(10.4)/x64(10.5-10.8) |
| Inception | Inspired by MINIX (a Unix-like system) and eventually after adding many features of GUI, Drivers etc, Linus Torvalds developed the framework of the OS that became LINUX in 1992. The LINUX kernel was released on 17th September, 1991 | In 1969, it was developed by a group of AT&T employees at Bell Labs and Dennis Ritchie. It was written in “C” language and was designed to be a portable, multi-tasking and multi-user system in a time-sharing configuration. |

**Buildroot:**

Buildroot is a tool that simplifies and automates the process of building a complete Linux system for an embedded system, using cross-compilation. In order to achieve this, Buildroot is able to generate a cross-compilation toolchain, a root filesystem, a Linux kernel image and a bootloader for your target.

**Buildroot** is a set of [Makefiles](https://en.wikipedia.org/wiki/Makefile) and [patches](https://en.wikipedia.org/wiki/Patch_(Unix)) that simplifies and automates the process of building a complete and bootable [Linux](https://en.wikipedia.org/wiki/Linux) environment for an [embedded system](https://en.wikipedia.org/wiki/Embedded_system), while using [cross-compilation](https://en.wikipedia.org/wiki/Cross_compiler) to allow building for multiple target platforms on a single Linux-based development system. **Buildroot can automatically build the required cross-compilation** [**toolchain**](https://en.wikipedia.org/wiki/Toolchain)**, create a** [**root file system**](https://en.wikipedia.org/wiki/Root_file_system)**, compile a** [**Linux kernel**](https://en.wikipedia.org/wiki/Linux_kernel) **image, and generate a** [**boot loader**](https://en.wikipedia.org/wiki/Boot_loader) **for the targeted embedded system,** **or it can perform any independent combination of these steps.** For example, an already installed cross-compilation toolchain can be used independently, while Buildroot only creates the root file system.

Buildroot is primarily intended to be used with small or embedded systems based on various [computer architectures](https://en.wikipedia.org/wiki/Computer_architectures) and [instruction set architectures](https://en.wikipedia.org/wiki/Instruction_set_architecture) (ISAs), including [x86](https://en.wikipedia.org/wiki/X86), [ARM](https://en.wikipedia.org/wiki/ARM_architecture), [MIPS](https://en.wikipedia.org/wiki/MIPS_architecture) and [PowerPC](https://en.wikipedia.org/wiki/PowerPC).[[6]](https://en.wikipedia.org/wiki/Buildroot" \l "cite_note-buildroot-manual-6):2 Numerous architectures and their variants are supported; Buildroot also comes with default configurations for several off-the-shelf available embedded boards, such as [Cubieboard](https://en.wikipedia.org/wiki/Cubieboard), [Raspberry Pi](https://en.wikipedia.org/wiki/Raspberry_Pi) and [SheevaPlug](https://en.wikipedia.org/wiki/SheevaPlug).[[8]](https://en.wikipedia.org/wiki/Buildroot" \l "cite_note-8)[[9]](https://en.wikipedia.org/wiki/Buildroot" \l "cite_note-elinux-elce2013-9):25 Several third-party projects and products use Buildroot as the basis for their build systems, including the [OpenWrt](https://en.wikipedia.org/wiki/OpenWrt) project that creates an [embedded operating system](https://en.wikipedia.org/wiki/Embedded_operating_system), and [firmware](https://en.wikipedia.org/wiki/Firmware) for the [customer-premises equipment](https://en.wikipedia.org/wiki/Customer-premises_equipment) (CPE) used by the [Google Fiber](https://en.wikipedia.org/wiki/Google_Fiber) broadband service.

**Translation Lookaside Buffer (TLB) :**

A translation lookaside buffer (**TLB**) is a memory cache that is used to reduce the time taken to access a user memory location. It is a part of the chip's memory-management unit (MMU). The **TLB** stores the recent translations of virtual memory to physical memory and can be called an address-translation cache.

Initially, when the processor needs to map a virtual address to a physical address, it must traverse the full page directory searching for the PTE of interest. This would normally imply that each assembly instruction that references memory actually requires several separate memory references for the page table traversal [[Tan01](https://www.kernel.org/doc/gorman/html/understand/understand031.html" \l "tanenbaum01)]. To avoid this considerable overhead, architectures take advantage of the fact that most processes exhibit a locality of reference or, in other words, large numbers of memory references tend to be for a small number of pages. They take advantage of this reference locality by providing a Translation Lookaside Buffer (TLB) which is a small associative memory that caches virtual to physical page table resolutions.

Linux assumes that the most architectures support some type of TLB although the architecture independent code does not cares how it works. Instead, architecture dependant hooks are dispersed throughout the VM code at points where it is known that some hardware with a TLB would need to perform a TLB related operation. For example, when the page tables have been updated, such as after a page fault has completed, the processor may need to be update the TLB for that virtual address mapping.

Not all architectures require these type of operations but because some do, the hooks have to exist. If the architecture does not require the operation to be performed, the function for that TLB operation will a null operation that is optimised out at compile time.

**Pages and Paging:**

Memory is composed of bits, of which (usually) eight make a byte. Bytes compose words, which in turn compose pages. For the purposes of memory management, the page is the most important of these: it is the smallest addressable unit of memory that thememory management unit (MMU) can manage. Thus the virtual address space is carved up into pages. The machine architecture determines the page size. Typical sizes include 4 KB for 32-bit systems and 8 KB for 64-bit systems.A process cannot necessarily access all of those pages; they may not correspond to anything. Thus, pages are either valid or invalid. A valid page is associated with an actual page of data, either in physical memory (RAM) or on secondary storage, such as a swap partition or file on disk. An invalid page is not associated with anything and represents an unused, unallocated piece

of the address space. Accessing an invalid page results in a segmentation violation.

If a valid page is associated with data on secondary storage, a process cannot access that

page until the data is brought into physical memory. When a process attempts to access

such a page, the memory management unit generates a page fault. The kernel then

intervenes, transparently paging in the data from secondary storage to physical memory.

Because there is considerably more virtual memory than physical memory, the kernel

may have to move data out of memory to make room for the data paging in. Paging

out is the process of moving data from physical memory to secondary storage. To min‐

imize subsequent page ins, the kernel attempts to page out the data that is the least likely

to be used in the near future.

# **Reentrant Function**

A function is said to be reentrant if there is a provision to interrupt the function in the course of execution, service the interrupt service routine and then resume the earlier going on function, without hampering its earlier course of action. Reentrant functions are used in applications like hardware interrupt handling, recursion, etc.  
The function has to satisfy certain conditions to be called as reentrant:  
1. It may not use global and static data. Though there are no restrictions, but it is generally not advised. because the interrupt may change certain global values and resuming the course of action of the reentrant function with the new data may give undesired results.  
2. It should not modify it’s own code. This is important because the course of action of the function should remain the same throughout the code. But, this may be allowed in case the interrupt routine uses a local copy of the reentrant function every time it uses different values or before and after the interrupt.  
3. Should not call another non-reentrant function.

**Pseudo Code**

Pseudo code is a term which is often used in programming and algorithm based fields. It is a methodology that allows the programmer to represent the implementation of an algorithm. Simply, we can say that it’s the cooked up representation of an algorithm. Often at times, algorithms are represented with the help of pseudo codes as they can be interpreted by programmers no matter what their programming background or knowledge is. Pseudo code, as the name suggests, is a false code or a representation of code which can be understood by even a layman with some school level programming knowledge.

[Algorithm](https://www.geeksforgeeks.org/fundamentals-of-algorithms/): It’s an organized logical sequence of the actions or the approach towards a particular problem. A programmer implements an algorithm to solve a problem. Algorithms are expressed using natural verbal but somewhat technical annotations.

Pseudo code: It’s simply an implementation of an algorithm in the form of annotations and informative text written in plain English. It has no syntax like any of the programming language and thus can’t be compiled or interpreted by the computer.

### Advantages of Pseudocode

* Improves the readability of any approach. It’s one of the best approaches to start implementation of an algorithm.
* Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that’s where a pseudo-code proves vital.
* The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

**Top and Bottom Halves**

One of the main problems with interrupt handling is how to perform lengthy tasks within a handler. Often a substantial amount of workmust be done in response to a device interrupt, but interrupt handlers need to finish up quickly and not keep interrupts blocked for long. These two needs (work and speed) conflict with each other, leaving the driver writer in a bit of a bind. Linux (along with many other systems) resolves this problem by splitting the interrupt handler into two halves. The so-called top half is the routine that actually responds to the interrupt—the one you register with request\_irq. The bottom half is a routine that is scheduled by the top half to be executed later, at a safer time. The big difference between the top-half handler and the bottom half is that all interrupts are enabled during execution of the bottom half—that’s why it runs at a safer time. In the typical scenario, the top half saves device data to a device-specific buffer, schedules its bottom half, and exits: this operation is very fast. The bottom half then performs whatever other work is required, such as awakening processes, starting up another I/O operation, and so on. This setup permits the top half to service a new interrupt while the bottom half is still working. Almost every serious interrupt handler is split this way. For instance, when a network interface reports the arrival of a new packet, the handler just retrieves the data and pushes it up to the protocol layer; actual processing of the packet is performed in a bottom half.

**Major and Minor Numbers:**

* Traditionally, the major number identifies the driver associated with the device.
* The minor number is used by the kernel to determine exactly which device is being referred to.
* Kernel versions were limited to 255 major and 255 minor numbers.