ACKNOWLEGEMENT

It is our solemn duty to place on record my sincere thanks and deep sense of gratitude towards my respected teacher, **Dr. Akshi Kumar** , for her guidance, motivation and constant encouragement for fulfilment of this project.

We would also like to thank the **Delhi Technological University** and all the concerned authorities for providing us this platform and giving us correct guidance and resources to complete our project in the most feasible way possible.

This project has been a great source of real life learning and it was more of a reality check rather than just a theory project. It was a great experience to know more about reality and how things work in real life.

ABSTRACT

Linux is a variant of UNIX that has gained popularity over the last several decades, powering devices as small as mobile phones and as large as roomfilling supercomputers.we look at the history and development of Linux and cover the user and programmer interfaces that Linux presents —interfaces that owe a great deal to the UNIX tradition. We also discuss the design and implementation of these interfaces. Linux is a rapidly evolving operating system.

Linux is an open-source operating system like other operating systems such as Microsoft Windows, Apple Mac OS, iOS, Google android, etc. An operating system is a software that enables the communication between computer hardware and software. It conveys input to get processed by the processor and brings output to the hardware to display it. This is the basic function of an operating system. Although it performs many other important tasks, let's not talk about that.

Linux is around us since the mid-90s. It can be used from wristwatches to supercomputers. It is everywhere in our phones, laptops, PCs, cars and even in refrigerators. It is very much famous among developers and normal computer users.

TABLE OF CONTENT

INTRODUCTION :-

Linux is a community of open-source Unix like operating systems that are based on the Linux Kernel. It was initially released by **Linus Torvalds** on September 17, 1991. It is a free and open-source operating system and the source code can be modified and distributed to anyone commercially or noncommercially under the GNU General Public License.

Initially, Linux was created for personal computers and gradually it was used in other machines like servers, mainframe computers, supercomputers, etc. Nowadays, Linux is also used in embedded systems like routers, automation controls, televisions, digital video recorders, video game consoles, smartwatches, etc. The biggest success of Linux is Android(operating system) it is based on the Linux kernel that is running on smartphones and tablets. Due to android Linux has the largest installed base of all general-purpose operating systems. Linux is generally packaged in a Linux distribution.

HISTORY OF LINUX:-

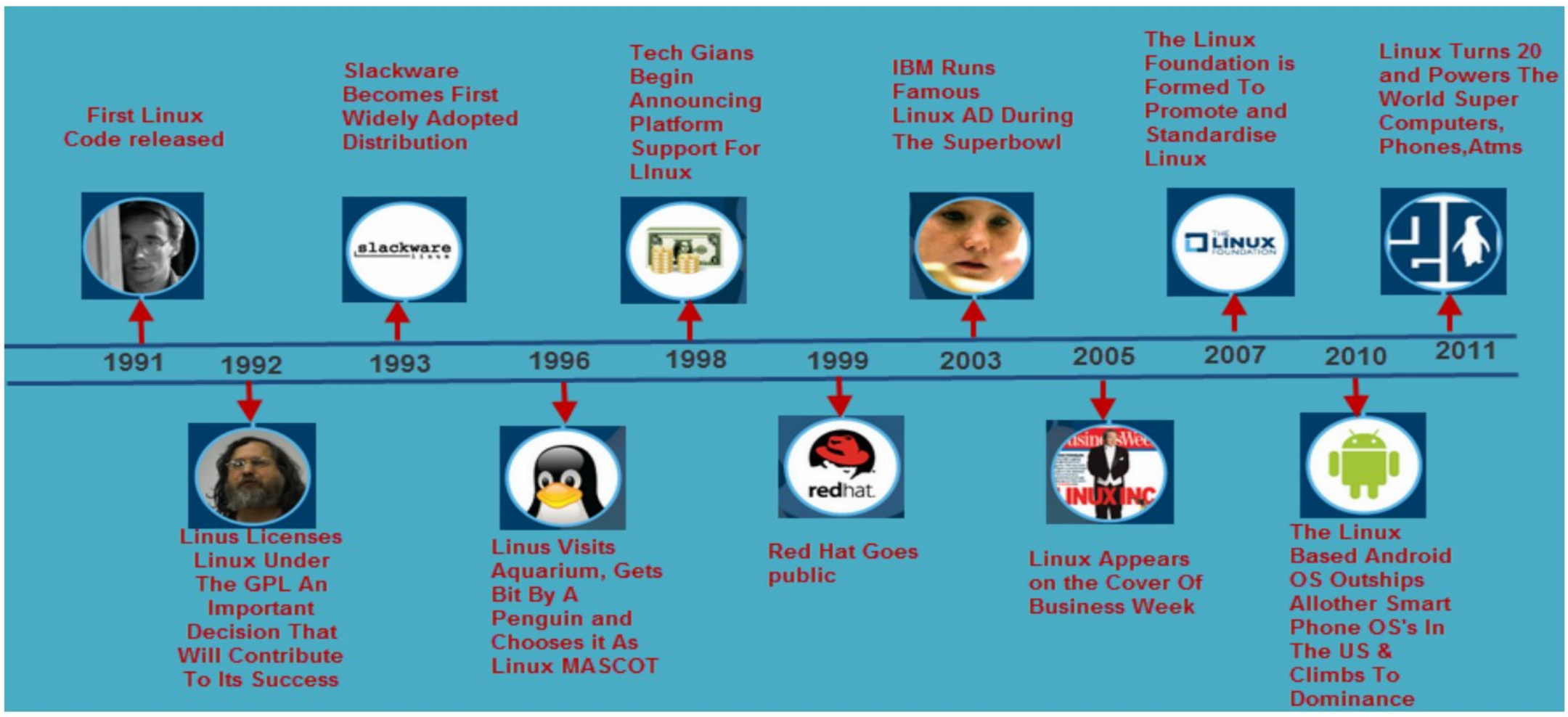
The History of Linux began in the 1991 with the beginning of a personal project by a Finland student Linus Torvalds to create a new free operating system kernel. Since then, the resulting Linux Kernel has been marked by constant growth throughout the history.

- In the year 1991, Linux was introduced by a Finland student Linus Torvalds.

- Hewlett Packard UniX(HP-UX) 8.0 was released.

- In the year 1992, Hewlett Packard 9.0 was released.

- In the year 1993, NetBSD 0.8 and FreeBSD 1.0 released.
- In the year 1994, Red Hat Linux was introduced, Caldera was founded by Bryan Sparks and Ransom Love and NetBSD1.0 Released.
- In the year 1995, FreeBSD 2.0 and HP UX 10.0 was released.
- In the year 1996, K Desktop Environment was developed by Matthias Ettrich.
- In the year 1997, HP-UX 11.0 was released.
- In the year 1998, the fifth generation of SGI Unix i.e IRIX 6.5 , Sun Solaris 7 operating system and Free BSD 3.0 was released.
- In the year 2000, the agreement of Caldera Systems with SCO server software division and the professional services division was announced.
- In the year 2001, Linus Torvalds released the Linux 2.4 version source code.
- In the year 2001, Microsoft filed a trademark suit against Lindows.com
- In the year 2004, Lindows name was changed to Linspire.
- In the year 2004, the first release of Ubuntu was released.
- In the year 2005, The project, openSUSE began a free distribution from Novell's community.
- In the year 2006, Oracle released its own distribution of Red Hat.
- In the year 2007, Dell started distributing laptops with Ubuntu pre installed in it.
- In the year 2011, Linux kernel 3.0 version was released.
- In the year 2013, Googles Linux based Android claimed 75% of the smartphone market share, in terms of the number of phones shipped.
- In the year 2014, Ubuntu claimed 22,000,000 users.



The History of Linux

WHAT IS LINUX OPERATING SYSTEM?

Just like Windows XP, Windows 7, Windows 8, and Mac OS X, Linux is an operating system. An operating system is software that manages all of the hardware resources associated with your desktop or laptop. To put it simply – the operating system manages the communication between your software and your hardware. Without the operating system (often referred to as the "OS"), the software wouldn't function.

The OS is comprised of a number of pieces:

- The Bootloader: The software that manages the boot process of your computer. For most users, this will simply be a splash screen that pops up and eventually goes away to boot into the operating system.

- The kernel: This is the one piece of the whole that is actually called "Linux". The kernel is the core of the system and manages the CPU, memory, and peripheral devices. The kernel is the "lowest" level of the OS.

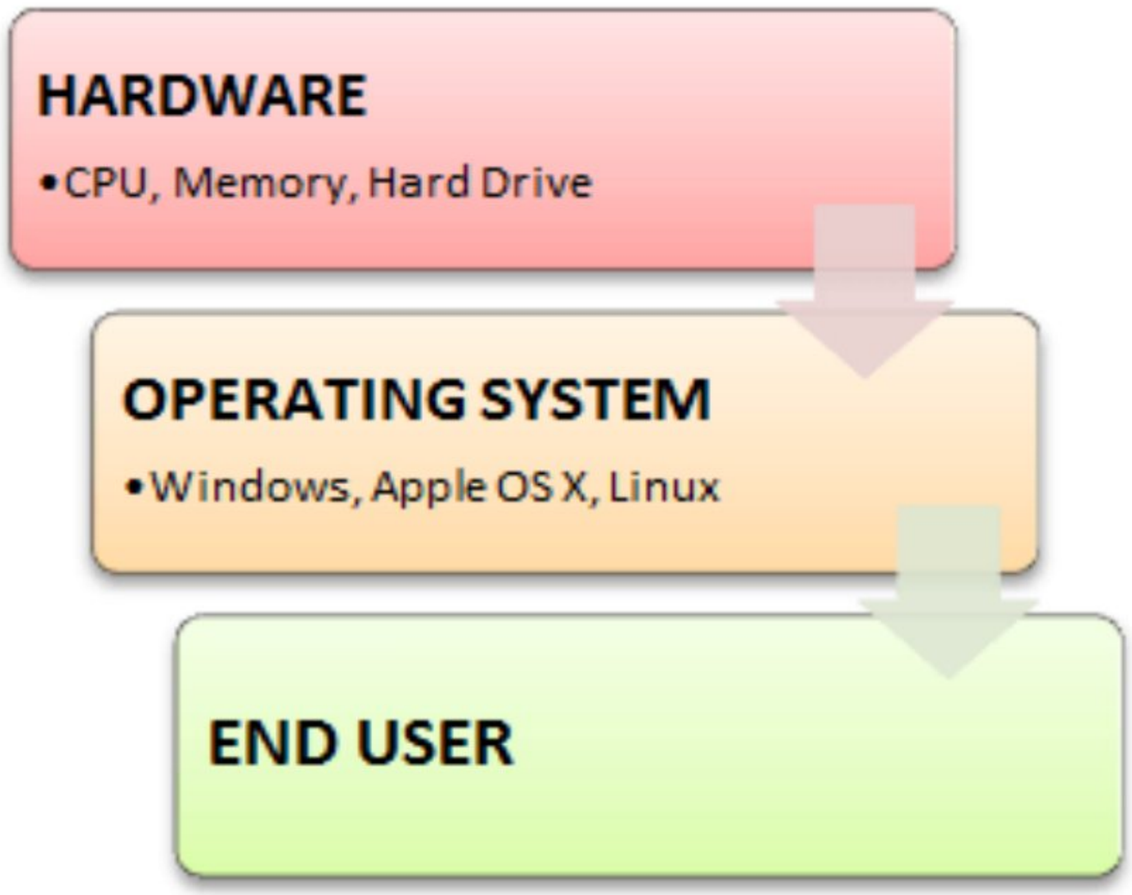
• Daemons: These are background services (printing, sound, scheduling, etc) that either start up during boot, or after you log into the desktop.

• The Shell: You've probably heard mention of the Linux command line. This is the shell – a command process that allows you to control the computer via commands typed into a text interface. This is what, at one time, scared people away from Linux the most (assuming they had to learn a seemingly archaic command line structure to make Linux work). This is no longer the case. With modern desktop Linux, there is no need to ever touch the command line.

• Graphical Server: This is the sub-system that displays the graphics on your monitor. It is commonly referred to as the X server or just "X".

• Desktop Environment: This is the piece of the puzzle that the users actually interact with. There are many desktop environments to choose from (Unity, GNOME, Cinnamon, Enlightenment, KDE, XFCE, etc). Each desktop environment includes built-in applications (such as file managers, configuration tools, web browsers, games, etc).

• Applications: Desktop environments do not offer the full array of apps. Just like Windows and Mac, Linux offers thousands upon thousands of high-quality software titles that can be easily found and installed. Most modern Linux distributions (more on this in a moment) include App Store-like tools that centralize and simplify application installation. For example: Ubuntu Linux has the Ubuntu Software Center which allows you to quickly search among the thousands of apps and install them from one centralized location



DESIGN GOALS OF LINUX:-

• UNIX was always an interactive system designed to handle multiple processes and multiple users at the same time. It was designed by programmers, for programmers, to use in an environment in which the majority of the users are relatively sophisticated and are engaged in (often quite complex) software development projects. In many cases, a large number of programmers are actively cooperating to produce a single system, so UNIX has extensive facilities to allow people to work together and share information in controlled ways.

• Linux is a multi-user, multitasking system with a full set of UNIX-compatible tools..

• Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model. • Main design goals are speed, efficiency, and standardization.

• Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification.

• The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behavior.

COMPONENTS OF LINUX SYSTEM:-

Linux Operating System has primarily three components

• Kernel − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.

• System Library − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.

• System Utility − System Utility programs are responsible to do specialized, individual level tasks.

Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called kernel mode with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in User Mode which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

Following are some of the important features of Linux Operating System.

• Portable − Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.

• Open Source − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

• Multi-User − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

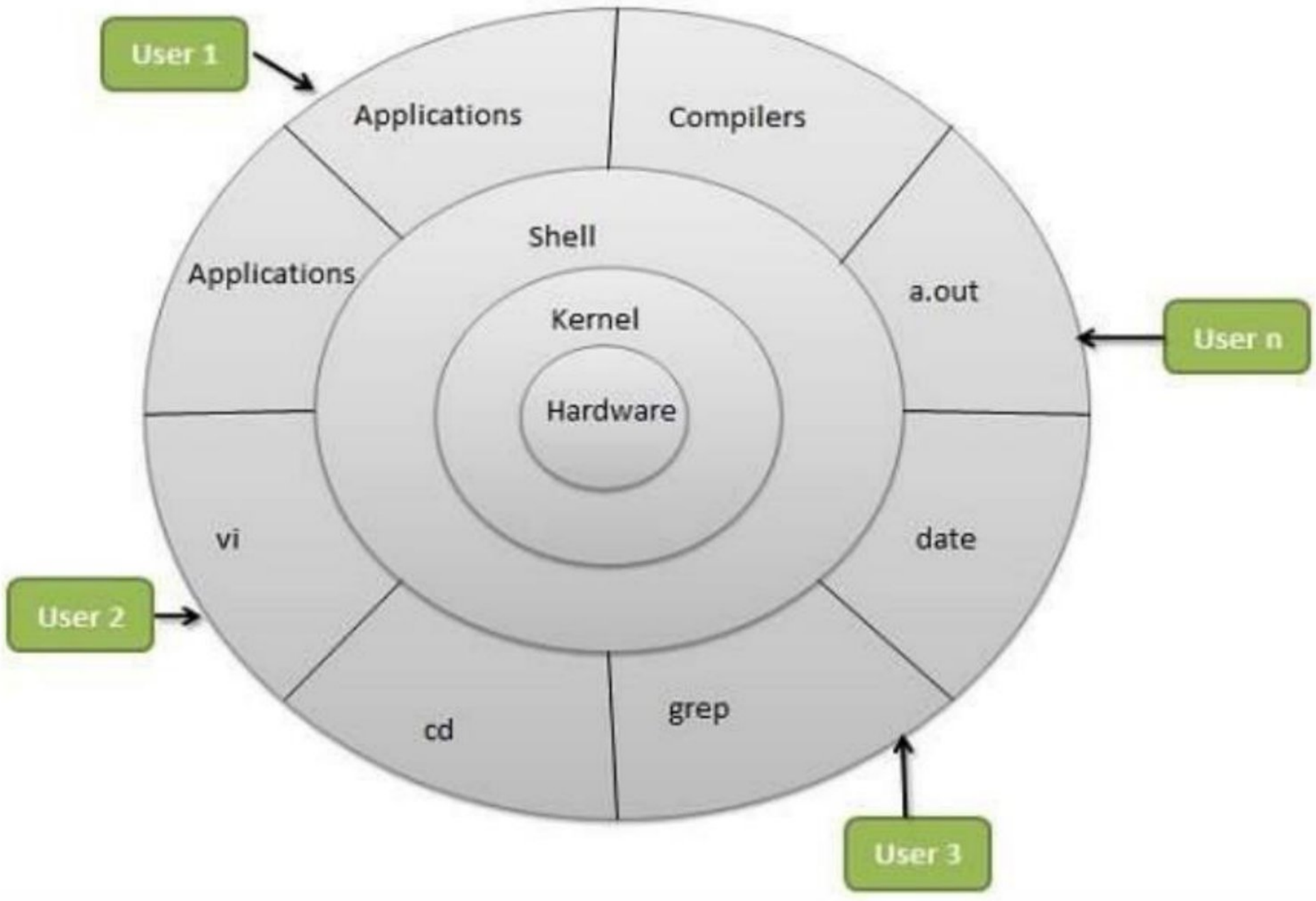
• Multiprogramming − Linux is a multiprogramming system means multiple applications can run at same time.

• Hierarchical File System − Linux provides a standard file structure in which system files/ user files are arranged.

• Shell − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.

• Security − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data. Architecture

The following illustration shows the architecture of a Linux system –



The architecture of a Linux System consists of the following layers –

• Hardware layer − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).

• Kernel − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.

• Shell − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions. • Utilities − Utility programs that provide the user most of the functionalities of an operating systems

ADVANTAGES OF LINUX:-

1. Open Source-

One of the main advantages of Linux is that it is an open source operating system i.e. its source code is easily available for everyone. Anyone capable of coding can contribute, modify, enhance and distribute the code to anyone and for any purpose.

2. Security-

Linux is more secure in comparison to other operating systems such as Windows. Linux is not completely secure as there is some malware for it also but it is less vulnerable than others. Every program in Linux whether an application or a virus needs authorization from the administrator in the form of a password. Unless the password is typed virus won't execute. There is no requirement of any anti-virus program in Linux.

3. Revive older computer systems-

Linux helps us to use or utilize old and outdated computer systems as a firewall, router, backup server or file server and many more. There are many distributions available to use according to your system capability. As we can use Puppy Linux for low- end systems.

4. Software Updates-

In Linux we encounter a larger number of software updates. These software updates are much faster than updates in any other operating system. Updates in Linux can be done easily without facing any major issue or concern.

5. Customization-

A feature that gives a major advantage over other operating systems is customization. You can customize any feature, add or delete any

feature according to your need as it is an open source operating system. Not only this, various wallpapers and attractive icon themes can be installed to give an amazing look to your system.

6. Various Distributions-

There are many distributions available also called distros of Linux. It provides various choices or flavors to the users. we can select any bistros according to your needs. Some bistros of Linux are Fedora, Ubuntu, Arch Linux, Debian, Linux Mint and many more. If we are a beginner we can use Ubuntu or Linux Mint. If you are a good programmer we may use Debian or Fedora.

7. Free to use (Low Cost)-

Linux is freely available on the web to download and use. We do not need to buy the license for it as Linux and many of its software come with GNU General Public License. This proved to be one of the major advantages Linux faces over Windows and other operating systems. We need to spend a huge amount to buy the license of Windows which is not the case with Linux.

8. Large Community Support-

Forums by excited users are made on the web to help and solve the problem any other user is facing. There are a lot of dedicated programmers there to help you out whenever and wherever possible.

9. Stability (Reliability)-

Linux provides high stability also this is good advantage i.e. it does not need to be rebooted after a short period of time. Linux system rarely slows down or freezes. As in windows, we need to reboot your system after installing or uninstalling an application or updating our software but this is not the case with Linux. we can work without any disturbance on our Linux systems.

10. Privacy-

Linux ensures the privacy of user's data as it never collects much data from the user while using its distributions or software but this is not true for many other operating systems.

11. Performance-

Linux provides high performance on various networks and workstations. It allows a large number of users to work simultaneously and handles them efficiently.

12. Network Support-

Linux gives support for network functionality as it was written by programmers over the internet. Linux helps you to set up client and server systems on your computer systems easily and in a fast manner.

13. Flexibility-

Linux provides a high range of flexibility as we can install only required components. There is no need to install a full or complete suite. We can also keep Linux file under multiple partitions so if one of them corrupts then there is no major loss. We only need to repair that particular partition, not the complete file which is not the case with other operating systems.

14. Compatibility-

Linux runs or executes all possible file formats and is compatible with a large number of file formats.

15. Fast and easy installation-

Linux can be easily installed from the web and does not require any prerequisites as it can run on any hardware, even on your oldest systems.

16. Proper use of Hard Disk-

Linux performs all the tasks efficiently even after the hard disk is almost full. This increases the performance of the Linux hence Linux provides high performance also.

17. Multitasking-

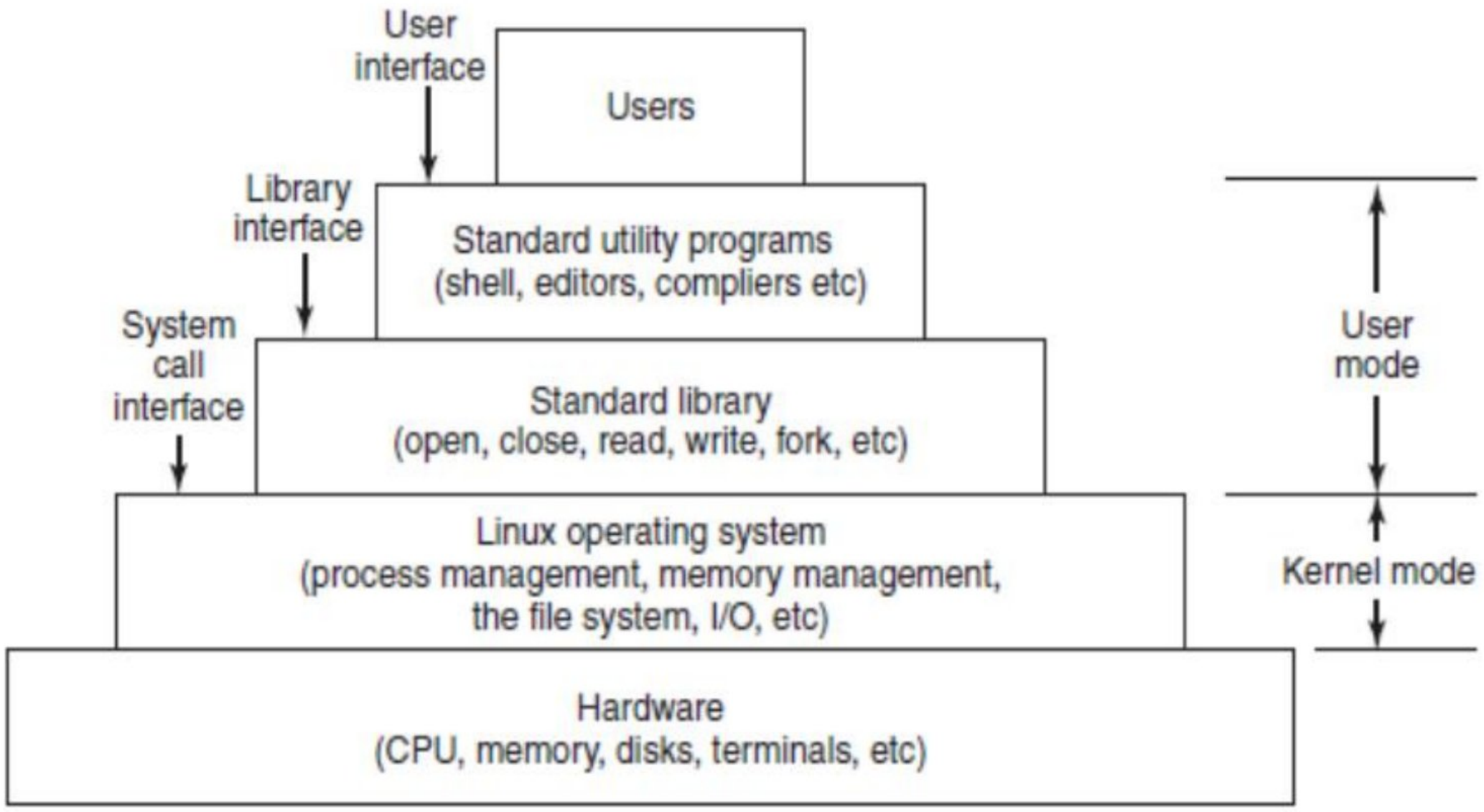
Linux is a multitasking operating system as it can perform many tasks simultaneously without any decrease in its speed such as downloading a large file would not slow down the system.

18. Run multiple desktops-

Linux provides various desktop environments to make it easy to use. While installing Linux you can choose any desktop environment according to your wishes such as KDE (K Desktop Environment) or GNOME (GNU Network Object Model Environment).

INTERFACES TO LINUX SYSTEM:-

. • A Linux system can be regarded as a kind of pyramid, as illustrated in Fig. At the bottom is the hardware, consisting of the CPU, memory, disks, a monitor and keyboard, and other devices. Running on the bare hardware is the operating system. Its function is to control the hardware and provide a system call interface to all the programs. These system calls allow user programs to create and manage processes, files, and other resources.



A Linux operating system can be divided in to the following layers:

1) **Hardware:** This is the bottom most layer of a Linux system. It consists of monitor, CPU, memory, disks, terminals, keyboards, and other devices.

2) **Linux operating system:** Linux operating system runs on the hardware. It controls the hardware and manages memory, processes, file systems, and Input/Output. It also provides a system call interface for the programs.

3) **System library:** This is the standard library for calling specific procedures. It provides a library interface for the system calls. It has various library procedures like read, write, fork, etc.

4) **Utility programs:** A Linux system has several standard utility programs like compilers, shell, editors, file manipulation utilities, text processors, and other programs which can be called by the user. It provides a user interface for these programs.

5) **Users:** This is the topmost layer in the Linux operating system. It consists of the users of the Linux operating system.

WHAT IS MEAN BY SHELL IN LINUX? WHAT IS ITS USE?

• Computer understand the language of 0's and 1's called binary language. In early days of computing, instruction are provided using binary language, which is difficult for all of us, to read and write. So in Os there is special program called Shell. Shell accepts your instruction or commands in English (mostly) and if its a valid command, it is pass to kernel.

• Shell is a user program or it's environment provided for user interaction. Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

• Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

The Bourne shell (sh) is a shell, or command-line interpreter, for computer operating systems. The Bourne shell was the default shell for Unix Version 7. • Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. First released in 1989, it has been distributed widely as it is a default shell on the major Linux distributions and OS X.

• C shell is the UNIX shell (command execution program, often called a command interpreter) created by Bill Joy at the University of California at Berkeley as an alternative to UNIX's original shell, the

Bourne shell . These two UNIX shells, along with the Korn shell , are the three most commonly used shells.

• The Korn shell is the UNIX shell (command execution program, often called a command interpreter) that was developed by David Korn of Bell Labs as a comprehensive combined version of other major UNIX shells.

• Tcsh is an enhanced, but completely compatible version of the Berkeley UNIX C shell (csh). It is a command language interpreter usable both as an interactive login shell and a shell script command processor. It includes a command-line editor, programmable word completion, spelling correction, a history mechanism, job control and a C-like syntax.

Several shell available with Linux including:

Shell Name	Developed by	Where	Remark
BASH (Bourne-Again Shell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	--
TCSH	See the man page. Type $ man tcsh	--	TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

LINUX UTILITY PROGRAMS:-

• The command-line (shell) user interface to Linux consists of a large number of standard utility programs. Roughly speaking, these programs can be divided into six categories, as follows:

1. File and directory manipulation commands.

2. Filters.

3. Program development tools, such as editors and compilers.

4. Text processing.

5. System administration.

6. Miscellaneous.

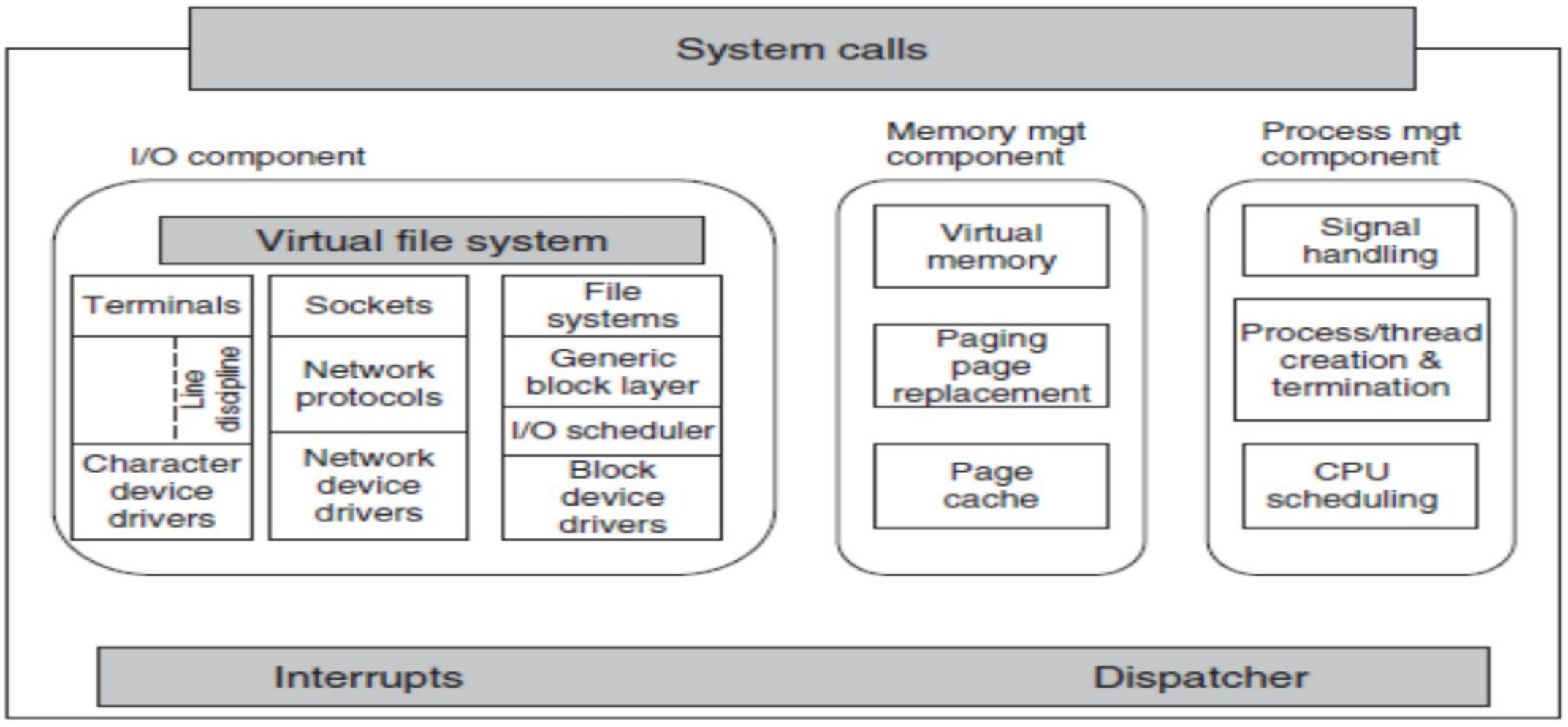
POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on the Unix operating system.

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

LINUX KERNEL WITH APPROPRIATE DIAGRAM:-

• The Linux kernel is a Unix-like computer operating system kernel. The Linux operating system is based on it and deployed on both traditional computer systems such as personal computers and servers, usually in the form of Linux distributions, and on various embedded devices such as routers, wireless access points, PBXes, set-top boxes, FTA receivers, smart TVs, PVRs and NAS appliances. The Android operating system for tablet computers, smartphones and smartwatches is also based atop the Linux kernel.

• The Linux kernel API, the application programming interface (API) through which user programs interact with the kernel, is meant to be very stable and to not break userspace programs (some programs, such as those with GUIs, rely on other APIs as well). As part of the kernel's functionality, device drivers control the hardware; "mainlined" device drivers are also meant to be very stable. However, the interface between the kernel and loadable kernel modules (LKMs), unlike in many other kernels and operating systems, is not meant to be very stable by design.



• The kernel sits directly on the hardware and enables interactions with I/O devices and the memory management unit and controls CPU access to them. At the lowest level, as shown in Fig. it contains interrupt handlers, which are the primary way for interacting with devices, and the low-level dispatching mechanism. This dispatching occurs when an interrupt happens. The low-level code here stops the running process, saves its state in the kernel process structures, and starts the appropriate driver. Process dispatching also happens when the kernel completes some operations and it is time to start up a user process again. The dispatching code is in assembler and is quite distinct from scheduling.

• Next, we divide the various kernel subsystems into three main components. The I/O component in Fig. contains all kernel pieces responsible for interacting with devices and performing network and storage I/O operations. At the highest level, the I/O operations are all integrated under a VFS (Virtual File System) layer. That is, at the top level, performing a read operation on a file, whether it is in memory or on disk, is the same as performing a read operation to retrieve a character from a terminal input. At the lowest level, all I/O operations pass through some device driver. All Linux drivers are classified as character-device drivers or block-device drivers, the main difference being that seeks and random accesses are allowed on block devices and not on character devices.

process in Linux and PID, UID, GID in Linux.

• Processes carry out tasks within the operating system. A program is a set of machine code instructions and data stored in an executable image on disk and is, as such, a passive entity; a process can be thought of as a computer program in action.

• During the lifetime of a process it will use many system resources. It will use the CPUs in the system to run its instructions and the system's physical memory to hold it and its data. It will open and use files within the file systems and may directly or indirectly use the

physical devices in the system. Linux must keep track of the process itself and of the system resources that it has so that it can manage it and the other processes in the system fairly. It would not be fair to the other processes in the system if one process monopolized most of the system's physical memory or its CPUs.

• The most precious resource in the system is the CPU, usually there is only one. Linux is a multiprocessing operating system, its objective is to have a process running on each CPU in the system at all times, to maximize CPU utilization. If there are more processes than CPUs (and there usually are), the rest of the processes must wait before a CPU becomes free until they can be run.

• Processes are created in Linux in an especially simple manner. The fork system call creates an exact copy of the original process. The forking process is called the parent process. The new process is called the child process. The parent and child each have their own, private memory images. If the parent subsequently changes any of its variables, the changes are not visible to the child, and vice versa.

• Process Identifier is when each process has a unique identifier associated with it known as process id.

• User and Group Identifiers (UID and GID) are the identifiers associated with a processes of the user and group.

• The new process are created by cloning old process or current process. A new task is created by a system call i.e fork or clone. The forking process is called parent process and the new process is called as child process.

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

LINUX PROCESSES:-

• **State:-** As a process executes it changes state according to its circumstances.

• **Running:-** The process is either running (it is the current process in the system) or it is ready to run (it is waiting to be assigned to one of the system's CPUs).

• **Waiting:-** The process is waiting for an event or for a resource. Linux differentiates between two types of waiting process; interruptible and uninterruptible. Interruptible waiting processes can be interrupted by signals whereas uninterruptible waiting processes are waiting directly on hardware conditions and cannot be interrupted under any circumstances.

• **Stopped:-** The process has been stopped, usually by receiving a signal. A process that is being debugged can be in a stopped state.

• **Zombie:-** This is a halted process which, for some reason, still has a data structure in the task vector. It is what it sounds like, a dead process.

• **Scheduling Information:-** The scheduler needs this information in order to fairly decide which process in the system most deserves to run,

• **Identifiers:-** Every process in the system has a process identifier. The process identifier is not an index into the task vector, it is simply a number. Each process also has User and group identifiers, these are used to control this processes access to the files and devices in the system,

• **Inter-Process Communication:-** Linux supports the classic Unix IPC mechanisms of signals, pipes and semaphores and also the System V IPC mechanisms of shared memory, semaphores and message queues.

• **Links:-** In a Linux system no process is independent of any other process. Every process in the system, except the initial process has a parent process. New processes are not created, they are copied, or rather cloned from previous processes

• **Times and Timers:-** The kernel keeps track of a processes creation time as well as the CPU time that it consumes during its lifetime. Each clock tick, the kernel updates the amount of time that the current process has spent in system and in user mode. Linux also supports process specific interval timers, processes can use system calls to set up timers to send signals to themselves when the timers expire. These timers can be singleshot or periodic timers.

• **Virtual memory:-** Most processes have some virtual memory (kernel threads and daemons do not) and the Linux kernel must track how that virtual memory is mapped onto the system's physical memory.

• **Processor Specific Context**:- A process could be thought of as the sum total of the system's current state. Whenever a process is running it is using the processor's registers, stacks and so on.

• **File system:-** Processes can open and close files as they wish and the processes contains pointers to descriptors for each open file as well as pointers to two VFS. Each VFS uniquely describes a file or directory within a file system and also provides a uniform interface to the underlying file systems.

PROCESS MANAGEMENT SYSTEM CALLS IN LINUX?

• Processes are the most fundamental abstraction in a Linux system, after files. As object code in execution - active, alive, running programs - processes are more than just assembly language; they consist of data, resources, state, and a virtualized computer.

• Linux took an interesting path, one seldom traveled, and separated the act of reating a new process from the act of loading a new binary image. Although the two tasks are performed in tandem most of the time, the division has allowed a great deal of freedom for experimentation and evolution for each of the tasks. This road less traveled has survived to this day, and while most operating systems offer a single system call to start up a new program, Linux requires two: a fork and an exec.

Pocess id

Syscall	Description
GETPID	Get process ID
GETPPID	Get parent process ID
GETTID	Get thread ID

Session id

Syscall	Description
SETSID	Set session ID
GETSID	Get session ID

Process group id

Syscall	Description
SETPGID	Set process group ID
GETPGID	Get process group ID
GETPGRP	Get the process group ID of the calling process

Users and groups

Syscall	Description
SETUID	Set real user ID
GETUID	Get real user ID
SETGID	Set real group ID
GETGID	Get real group ID
SETRESUID	Set real, effective and saved user IDs
GETRESUID	Get real, effective and saved user IDs
SETRESGID	Set real, effective and saved group IDs
GETRESGID	Get real, effective and saved group IDs
SETREUID	Set real and/or effective user ID
SETREGID	Set real and/or effective group ID
SETFSUID	Set user ID used for file system checks
SETFSGID	Set group ID used for file system checks
GETEUID	Get effective user ID

GETEGID	Get effective group ID
SETGROUPS	Set list of supplementary group IDs
GETGROUPS	Get list of supplementary group IDs

MAKING A MINIMUL LINUX FROM SCRATCH :

Download any of the most common distributions to install and configure—be it Ubuntu, Debian, Fedora, OpenSUSE etc. Building your own custom you can use minimal Linux distribution.

- **Host:** system work, including cross compilation and installation of the target image.
- **Target:** the *target* is the final cross-compiled operating system that you'll be building from source packages. It'll be built using the cross compiler on the host machine.
- **Cross compiler:** you'll be building and using a *cross compiler* to create the *target* image on the *host* machine. A cross compiler is built to run on a host machine, but it's used to compile for a target architecture or microprocessor that isn't compatible with the host machine.

PREREQUISITES AND TOOLS

The GCC, make, ncurses, Perl and grub tools (specifically grub-install) should be installed on the host machine.

You need to download and build all the packages for the cross compiler and the target image. I'm using the following open-source packages and versions for this tutorial:

- binutils-2.30.tar.xz
- busybox-1.28.3.tar.bz2
- clfs-embedded-bootscripts-1.0-pre5.tar.bz2
- gcc-7.3.0.tar.xz
- glibc-2.27.tar.xz
- gmp-6.1.2.tar.bz2
- linux-4.16.3.tar.xz
- mpc-1.1.0.tar.gz
- mpfr-4.0.1.tar.xz
- zlib-1.2.11.tar.gz

Configuring the Environment

First, turn on Bash hash functions:

```
$ set +h
```

Make sure that newly created files/directories are writable only by the owner:

```
$ umask 022
```

You'll use your home directory as the main build directory. (this isn't a requirement). This is where the cross-compilation tool chain and target image will be installed and put into the lj-os subdirectory. If you prefer to install it elsewhere, make the adjustment to the code section below:

```
$ export LJOS=~/lj-os
$ mkdir -pv ${LJOS}
```

Finally, export some remaining variables:

```
$ export LC_ALL=POSIX
$ export PATH=${LJOS}/cross-tools/bin:/bin:/usr/bin
```

After setting the above environment variables, create the target image's filesystem hierarchy:

```
$ mkdir -pv
${LJOS}/{bin,boot{,grub},dev,{etc/,}opt,home,
↪lib/{firmware,modules},lib64,mnt}
$ mkdir -pv
${LJOS}/{proc,media/{floppy,cdrom},sbin,srv,sys}
$ mkdir -pv ${LJOS}/var/{lock,log,mail,run,spool}
$ mkdir -pv
${LJOS}/var/{opt,cache,lib/{misc,locate},local}
$ install -dv -m 0750 ${LJOS}/root
$ install -dv -m 1777 ${LJOS}{/var,}/tmp
```

```
$ install -dv ${LJOS}/etc/init.d
$ mkdir -pv
${LJOS}/usr/{,local/}{bin,include,lib{,64},sbin,src}
$ mkdir -pv
${LJOS}/usr/{,local/}share/{doc,info,locale,man}
$ mkdir -pv
${LJOS}/usr/{,local/}share/{misc,terminfo,zoneinfo}
$ mkdir -pv
${LJOS}/usr/{,local/}share/man/man{1,2,3,4,5,6,7,8}
$ for dir in ${LJOS}/usr{,/local}; do
     ln -sv share/{man,doc,info} ${dir}
   done
```

This directory tree is based on the File system Hierarchy Standard (FHS), which is defined and hosted by the Linux Foundation:

Create the directory for a cross-compilation tool chain:

```
$ install -dv ${LJOS}/cross-tools{,/bin}
```

Use a symlink to /proc/mounts to maintain a list of mounted file systems properly in the /etc/mtab file:

```
$ ln -svf ../proc/mounts ${LJOS}/etc/mtab
```

Then create the /etc/passwd file, listing the root user account:

```
$ cat > ${LJOS}/etc/passwd << "EOF"
root::0:0:root:/root:/bin/ash
EOF
```

Create the /etc/group file with the following command:

```
$ cat > ${LJOS}/etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
```

```
kmem:x:3:
tty:x:4:
daemon:x:6:
disk:x:8:
dialout:x:10:
video:x:12:
utmp:x:13:
usb:x:14:
EOF
```

The target system's /etc/fstab:

```
$ cat > ${LJOS}/etc/fstab << "EOF"
# file system  mount-point  type   options
dump  fsck
#
order

rootfs           /               auto    defaults
1      1
proc             /proc           proc    defaults
0      0
sysfs            /sys            sysfs   defaults
0      0
devpts           /dev/pts        devpts
gid=4,mode=620   0      0
tmpfs            /dev/shm        tmpfs   defaults
0      0
EOF
```

The target system's /etc/profile to be used by the Almquist shell (ash) once the user is logged in to the target machine:

```
$ cat > ${LJOS}/etc/profile << "EOF"
export PATH=/bin:/usr/bin

if [ `id -u` -eq 0 ] ; then
        PATH=/bin:/sbin:/usr/bin:/usr/sbin
        unset HISTFILE
fi
```

```
# Set up some environment variables.
export USER=`id -un`
export LOGNAME=$USER
export HOSTNAME=`/bin/hostname`
export HISTSIZE=1000
export HISTFILESIZE=1000
export PAGER='/bin/more '
export EDITOR='/bin/vi'
EOF
```

The target machine's hostname (you can change this any time):

```
$ echo "ljos-test" > ${LJOS}/etc/HOSTNAME
```

And, /etc/issue, which will be displayed prominently at the login prompt:

```
$ cat > ${LJOS}/etc/issue<< "EOF"
Linux Journal OS 0.1a
Kernel \r on an \m

EOF
```

You won't use systemd here (this wasn't a political decision; it's due to convenience and for simplicity's sake). Instead, you'll use the basic `init` process provided by BusyBox. This requires that you define an /etc/inittab file:

```
$ cat > ${LJOS}/etc/inittab<< "EOF"
::sysinit:/etc/rc.d/startup

tty1::respawn:/sbin/getty 38400 tty1
tty2::respawn:/sbin/getty 38400 tty2
tty3::respawn:/sbin/getty 38400 tty3
tty4::respawn:/sbin/getty 38400 tty4
tty5::respawn:/sbin/getty 38400 tty5
tty6::respawn:/sbin/getty 38400 tty6
```

```
::shutdown:/etc/rc.d/shutdown
::ctrlaltdel:/sbin/reboot
EOF
```

Also as a result of leveraging BusyBox to simplify some of the most common Linux system functionality, you'll use `mdev` instead of `udev`, which requires you to define the following /etc/mdev.conf file:

```
$ cat > ${LJOS}/etc/mdev.conf<< "EOF"
# Devices:
# Syntax: %s %d:%d %s
# devices user:group mode

# null does already exist; therefore ownership has to
# be changed with command
null    root:root 0666  @chmod 666 $MDEV
zero    root:root 0666
grsec   root:root 0660
full    root:root 0666

random  root:root 0666
urandom root:root 0444
hwrandom root:root 0660

# console does already exist; therefore ownership has
to
# be changed with command
console root:tty 0600 @mkdir -pm 755 fd && cd fd &&
for x
 ↪in 0 1 2 3 ; do ln -sf /proc/self/fd/$x $x; done

kmem    root:root 0640
mem     root:root 0640
port    root:root 0640
ptmx    root:tty 0666

# ram.*
ram([0-9]*)     root:disk 0660 >rd/%1
loop([0-9]+)    root:disk 0660 >loop/%1
sd[a-z].*       root:disk 0660
*/lib/mdev/usbdisk_link
hd[a-z][0-9]*   root:disk 0660 */lib/mdev/ide_links
```

```
tty                 root:tty 0666
tty[0-9]            root:root 0600
tty[0-9][0-9]       root:tty 0660
ttyO[0-9]*          root:tty 0660
pty.*               root:tty 0660
vcs[0-9]*           root:tty 0660
vcsa[0-9]*          root:tty 0660

ttyLTM[0-9]         root:dialout 0660 @ln -sf $MDEV modem
ttySHSF[0-9]        root:dialout 0660 @ln -sf $MDEV modem
slamr               root:dialout 0660 @ln -sf $MDEV
slamr0
slusb               root:dialout 0660 @ln -sf $MDEV
slusb0
fuse                root:root  0666

# misc stuff
agpgart             root:root 0660  >misc/
psaux               root:root 0660  >misc/
rtc                 root:root 0664  >misc/

# input stuff
event[0-9]+         root:root 0640 =input/
ts[0-9]             root:root 0600 =input/

# v4l stuff
vbi[0-9]            root:video 0660 >v4l/
video[0-9]          root:video 0660 >v4l/

# load drivers for usb devices
usbdev[0-9].[0-9]           root:root 0660
*/lib/mdev/usbdev
usbdev[0-9].[0-9]_.*        root:root 0660
EOF
```

You'll need to create a /boot/grub/grub.cfg for the GRUB bootloader that will be installed on the target machine's physical or virtual HDD (note: the kernel image defined in this file needs to reflect the image built and installed on the target machine):

```
$ cat > ${LJOS}/boot/grub/grub.cfg<< "EOF"
```

```
set default=0
set timeout=5

set root=(hd0,1)

menuentry "Linux Journal OS 0.1a" {
        linux   /boot/vmlinuz-4.16.3 root=/dev/sda1
ro quiet
}
EOF
```

Finally, initialize the log files and give them proper permissions:

```
$ touch ${LJOS}/var/run/utmp
${LJOS}/var/log/{btmp,lastlog,wtmp}
$ chmod -v 664 ${LJOS}/var/run/utmp
${LJOS}/var/log/lastlog
```

Building the Cross Compiler

If you recall, the cross compiler is a tool chain of various compilation tools built for the system on which it's executing but designed to compile for an architecture or microprocessor that's not necessarily compatible with the system on which you're using it. In my environment, I'm running a 64-bit x86 architecture (x86-64) and will be cross compiling to a generic x86-64 target architecture. Sure, this section is somewhat redundant considering the environment I am running in, but the tutorial is designed to ensure that you are able to build for an x86-64 target, regardless of the machine type that you are using (for example, PowerPC, ARM, x86 and so on).

You never can be too sure with what is set in a currently running environment, which is why you'll unset the following C and C++ flags:

```
$ unset CFLAGS
$ unset CXXFLAGS
```

Next, define the most vital parts of the host/target variables needed to create the cross-compiler tool chain and target image:

```
$ export LJOS_HOST=$(echo ${MACHTYPE} | sed "s/-[^-
]*/-cross/")
$ export LJOS_TARGET=x86_64-unknown-linux-gnu
$ export LJOS_CPU=k8
$ export LJOS_ARCH=$(echo ${LJOS_TARGET} | sed -e
 ↪'s/-.*//' -e 's/i.86/i386/')
$ export LJOS_ENDIAN=little
```

Kernel Headers

The kernel's standard header files need to be installed for the cross compiler. Uncompress the kernel tarball and change into its directory. Then run:

```
$ make mrproper
$ make ARCH=${LJOS_ARCH} headers_check && \
make ARCH=${LJOS_ARCH} INSTALL_HDR_PATH=dest
headers_install
$ cp -rv dest/include/* ${LJOS}/usr/include
```

Binutils

Binutils contains a linker, assembler and other tools needed to handle compiled object files. Uncompress the tarball. Then create the binutils-build directory and change into it:

```
$ mkdir binutils-build
$ cd binutils-build/
```

Then run:

```
$ ../binutils-2.30/configure --prefix=${LJOS}/cross-
tools \
--target=${LJOS_TARGET} --with-sysroot=${LJOS} \
--disable-nls --enable-shared --disable-multilib
$ make configure-host && make
$ ln -sv lib ${LJOS}/cross-tools/lib64
$ make install
```

Copy over the following header file to the target's filesystem:

```
$ cp -v ../binutils-2.30/include/libiberty.h
${LJOS}/usr/include
```

GCC (Static)

Before building the final cross-compiler toolchain, you first must build a statically compiled toolchain to build the C library (glibc) to which the final GCC cross compiler will link.

Uncompress the GCC tarball, and then uncompress the following packages and move them into the GCC root directory:

```
$ tar xjf gmp-6.1.2.tar.bz2
$ mv gmp-6.1.2 gcc-7.3.0/gmp
$ tar xJf mpfr-4.0.1.tar.xz
$ mv mpfr-4.0.1 gcc-7.3.0/mpfr
$ tar xzf mpc-1.1.0.tar.gz
$ mv mpc-1.1.0 gcc-7.3.0/mpc
```

Now create a gcc-static directory and change into it:

```
$ mkdir gcc-static
$ cd gcc-static/
```

Run the following commands:

```
$ AR=ar LDFLAGS="-Wl,-rpath,${LJOS}/cross-tools/lib"
\
../gcc-7.3.0/configure --prefix=${LJOS}/cross-tools \
--build=${LJOS_HOST} --host=${LJOS_HOST} \
--target=${LJOS_TARGET} \
--with-sysroot=${LJOS}/target --disable-nls \
--disable-shared \
--with-mpfr-include=$(pwd)/../gcc-7.3.0/mpfr/src \
--with-mpfr-lib=$(pwd)/mpfr/src/.libs \
```

```
--without-headers --with-newlib --disable-decimal-
float \
--disable-libgomp --disable-libmudflap --disable-
libssp \
--disable-threads --enable-languages=c,c++ \
--disable-multilib --with-arch=${LJOS_CPU}
$ make all-gcc all-target-libgcc && \
make install-gcc install-target-libgcc
$ ln -vs libgcc.a `${LJOS_TARGET}-gcc -print-libgcc-
file-name |
 ↪sed 's/libgcc/&_eh/'`
```

Do not delete these directories; you'll need to come back to them from the final version of GCC.

Glibc

Uncompress the glibc tarball. Then create the glibc-build directory and change into it:

```
$ mkdir glibc-build
$ cd glibc-build/
```

Configure the following build flags:

```
$ echo "libc_cv_forced_unwind=yes" > config.cache
$ echo "libc_cv_c_cleanup=yes" >> config.cache
$ echo "libc_cv_ssp=no" >> config.cache
$ echo "libc_cv_ssp_strong=no" >> config.cache
```

Then run:

```
$ BUILD_CC="gcc" CC="${LJOS_TARGET}-gcc" \
AR="${LJOS_TARGET}-ar" \
RANLIB="${LJOS_TARGET}-ranlib" CFLAGS="-O2" \
../glibc-2.27/configure --prefix=/usr \
--host=${LJOS_TARGET} --build=${LJOS_HOST} \
--disable-profile --enable-add-ons --with-tls \
```

```
--enable-kernel=2.6.32 --with-__thread \
--with-binutils=${LJOS}/cross-tools/bin \
--with-headers=${LJOS}/usr/include \
--cache-file=config.cache
$ make && make install_root=${LJOS}/ install
```

GCC (Final)

As I mentioned previously, you'll now build the final GCC cross compiler that will link to the C library built and installed in the previous step. Create the gcc-build directory and change into it:

```
$ mkdir gcc-build
$ cd gcc-build/
```

Then run:

```
$ AR=ar LDFLAGS="-Wl,-rpath,${LJOS}/cross-tools/lib"
\
../gcc-7.3.0/configure --prefix=${LJOS}/cross-tools \
--build=${LJOS_HOST} --target=${LJOS_TARGET} \
--host=${LJOS_HOST} --with-sysroot=${LJOS} \
--disable-nls --enable-shared \
--enable-languages=c,c++ --enable-c99 \
--enable-long-long \
--with-mpfr-include=$(pwd)/../gcc-7.3.0/mpfr/src \
--with-mpfr-lib=$(pwd)/mpfr/src/.libs \
--disable-multilib --with-arch=${LJOS_CPU}
$ make && make install
$ cp -v ${LJOS}/cross-tools/${LJOS_TARGET}/lib64/
↪libgcc_s.so.1 ${LJOS}/lib64
```

Now that you've built the cross compiler, you need to adjust and export the following variables:

```
$ export CC="${LJOS_TARGET}-gcc"
$ export CXX="${LJOS_TARGET}-g++"
$ export CPP="${LJOS_TARGET}-gcc -E"
```

```
$ export AR="${LJOS_TARGET}-ar"
$ export AS="${LJOS_TARGET}-as"
$ export LD="${LJOS_TARGET}-ld"
$ export RANLIB="${LJOS_TARGET}-ranlib"
$ export READELF="${LJOS_TARGET}-readelf"
$ export STRIP="${LJOS_TARGET}-strip"
```

Building the Target Image

The hard part is now complete—you have the cross compiler. Now, let's focus on building the components that will be installed on the target image. This includes various libraries and utilities and, of course, the Linux kernel itself.

BusyBox

BusyBox is one of my all-time favorite open-source projects. The project advertises itself to be the Swiss Army knife of open-source utilities, and that's probably the best description one could give the project. BusyBox combines a large collection of tiny versions of the most commonly used Linux utilities into a single distributed package. Those tools range from common binaries, text editors and command-line shells to filesystem and networking utilities, process management tools and many more.

Uncompress the tarball and change into its directory. Then load the default compilation configuration template:

```
$ make CROSS_COMPILE="${LJOS_TARGET}-" defconfig
```

The default configuration template will enable the compilation of a default defined set of utilities and libraries. You can enable/disable whatever you see fit by running `menuconfig`:

```
$ make CROSS_COMPILE="${LJOS_TARGET}-" menuconfig
```

Compile and install the package:

```
$ make CROSS_COMPILE="${LJOS_TARGET}-"
$ make CROSS_COMPILE="${LJOS_TARGET}-" \
CONFIG_PREFIX="${LJOS}" install
```

Install the following Perl script, as you'll need it for the kernel build below:

```
$ cp -v examples/depmod.pl ${LJOS}/cross-tools/bin
$ chmod 755 ${LJOS}/cross-tools/bin/depmod.pl
```

The Linux Kernel

Change into the kernel package directory and run the following to set the default x86-64 configuration template:

```
$ make ARCH=${LJOS_ARCH} \
CROSS_COMPILE=${LJOS_TARGET}- x86_64_defconfig
```

This will define a minimum set of modules and settings for the compilation process. You most likely will need to make the proper adjustments for the target machine's environment. This includes enabling modules for storage and networking controllers and more. You can do that with the `menuconfig` option:

```
$ make ARCH=${LJOS_ARCH} \
CROSS_COMPILE=${LJOS_TARGET}- menuconfig
```

For instance, I'm going to be running this target image in a VirtualBox virtual machine where it will rely on an Intel `e1000` networking module (defaulted in defconfig) and an LSI `mpt2sas` storage controller for the operating system drive. For the sake of simplicity, these modules are configured to be compiled statically into the kernel image—that is, set to `*` instead of `m`. *Be sure to review what's needed and enable it, or your target environment will not operate properly when booted.*

Compile and install the kernel:

```
$ make ARCH=${LJOS_ARCH} \
CROSS_COMPILE=${LJOS_TARGET}-
$ make ARCH=${LJOS_ARCH} \
```

```
CROSS_COMPILE=${LJOS_TARGET}- \
INSTALL_MOD_PATH=${LJOS} modules_install
```

You'll need to copy a few files into the /boot directory for GRUB:

```
$ cp -v arch/x86/boot/bzImage ${LJOS}/boot/vmlinuz-
4.16.3
$ cp -v System.map ${LJOS}/boot/System.map-4.16.3
$ cp -v .config ${LJOS}/boot/config-4.16.3
```

Then run the previously installed Perl script provided by the BusyBox package:

```
$ ${LJOS}/cross-tools/bin/depmod.pl \
-F ${LJOS}/boot/System.map-4.16.3 \
-b ${LJOS}/lib/modules/4.16.3
```

The Bootscripts

The Cross Linux From Scratch (CLFS) project (a fork of the original LFS project) provides a wonderful set of bootscripts that I use here for simplicity's sake. Uncompress the package and change into its directory. Out of box, one of the package's makefiles contains a line that may not be compatible with your current working shell. Apply the following changes to the package's root Makefile to ensure that you don't experience any issues with package installation:

```
@@ -19,7 +19,9 @@ dist:
        rm -rf "dist/clfs-embedded-bootscripts-
$(VERSION)"

 create-dirs:
-       install -d -m ${DIRMODE}
 ↪${EXTDIR}/rc.d/{init.d,start,stop}
+       install -d -m ${DIRMODE}
${EXTDIR}/rc.d/init.d
+       install -d -m ${DIRMODE} ${EXTDIR}/rc.d/start
+       install -d -m ${DIRMODE} ${EXTDIR}/rc.d/stop
```

```
 install-bootscripts: create-dirs
        install -m ${CONFMODE}
clfs/rc.d/init.d/functions
         ↪${EXTDIR}/rc.d/init.d/
```

Then run the following commands to install and configure the target environment appropriately:

```
$ make DESTDIR=${LJOS}/ install-bootscripts
$ ln -sv ../rc.d/startup ${LJOS}/etc/init.d/rcS
```

Zlib

Now you're at the very last package for this tutorial. Zlib isn't a requirement, but it serves as a great guide for other packages you may want to install for your environment. Feel free to skip this step if you'd rather format and configure the physical or virtual HDD.

Uncompress the Zlib tarball and change into its directory. Then configure, build and install the package:

```
$ sed -i 's/-O3/-Os/g' configure
$ ./configure --prefix=/usr --shared
$ make && make DESTDIR=${LJOS}/ install
```

Now, because some packages may look for Zlib libraries in the /lib directory instead of the /lib64 directory, apply the following changes:

```
$ mv -v ${LJOS}/usr/lib/libz.so.* ${LJOS}/lib
$ ln -svf ../../lib/libz.so.1 ${LJOS}/usr/lib/libz.so
$ ln -svf ../lib/libz.so.1 ${LJOS}/lib64/libz.so.1
```

Installing the Target Image

All of the cross compilation is complete. Now you have everything you need to install the entire cross-compiled operating system to either a physical or virtual

drive, but before doing that, let's not tamper with the original target build directory by making a copy of it:

```
$ cp -rf ${LJOS}/ ${LJOS}-copy
```

Use this copy for the remainder of this tutorial. Remove some of the now unneeded directories:

```
$ rm -rfv ${LJOS}-copy/cross-tools
$ rm -rfv ${LJOS}-copy/usr/src/*
```

Followed by the now unneeded statically compiled library files (if any):

```
$ FILES="$(ls ${LJOS}-copy/usr/lib64/*.a)"
$ for file in $FILES; do
> rm -f $file
> done
```

Now strip all debug symbols from the installed binaries. This will reduce overall file sizes and keep the target image's overall footprint to a minimum:

```
$ find ${LJOS}-copy/{,usr/}{bin,lib,sbin} -type f
 ↪-exec sudo strip --strip-debug '{}' ';'
$ find ${LJOS}-copy/{,usr/}lib64 -type f -exec sudo
 ↪strip --strip-debug '{}' ';'
```

Finally, change file ownerships and create the following nodes:

```
$ sudo chown -R root:root ${LJOS}-copy
$ sudo chgrp 13 ${LJOS}-copy/var/run/utmp
 ↪${LJOS}-copy/var/log/lastlog
$ sudo mknod -m 0666 ${LJOS}-copy/dev/null c 1 3
$ sudo mknod -m 0600 ${LJOS}-copy/dev/console c 5 1
$ sudo chmod 4755 ${LJOS}-copy/bin/busybox
```

Change into the target copy directory to create a tarball of the entire operating system image:

```
$ cd {LJOS}-copy/
$ sudo tar cfJ ../ljos-build-21April2018.tar.xz *
```

Notice how the target image is less than 60MB. You built that—a minimal Linux operating system that occupies less than 60MB of disk space:

```
$ sudo du -h|tail -n1
58M     .
```

And, that same operating system compresses to less than 20MB:

```
$ ls -lh ljos-build-21April2018.tar.xz
-rw-r--r-- 1 root root 18M Apr 21 15:31
 ↪ljos-build-21April2018.tar.xz
```

For the rest of this tutorial, you'll need a disk drive. It will need to enumerate as a traditional block device (in my case, it's /dev/sdd):

```
$ cat /proc/partitions |grep sdd
   8       48     256000 sdd
```

That block device will need to be partitioned. A single partition should suffice, and you can use any one of a number of partition utilities, including `fdisk` or `parted`. Once that partition is created and detected by the host system, format the partition with an ext4 filesystem, mount that partition to a staging area and change into that directory:

```
$ sudo mkfs.ext4 /dev/sdd1
$ sudo mkdir tmp
$ sudo mount /dev/sdd1 tmp/
$ cd tmp/
```

Uncompress the operating system tarball of the entire target operating system into the root of the staging directory:

```
$ sudo tar xJf ../ljos-build-21April2018.tar.xz
```

Now run `grub-install` to install all the necessary modules and boot records to the volume:

```
$ sudo grub-install --root-
directory=/home/petros/tmp/ /dev/sdd
```

The `--root-directory` parameter defines the absolute path of the staging directory, while the last parameter is the block device without the partition's label.

Booting Up for the First Time

Now you're officially done. Install the HDD to the physical or virtual machine (as the primary disk drive) and power it up. You immediately will be greeted by the GRUB bootloader .

And within one second you'll be at the operating system's login prompt.

You'll notice a couple boot "error" and warning messages. This is because you're missing a couple files. You can correct that as you continue to learn the environment and build more packages into the operating system.

If you recall, you never set a root password. This was intentional. Log in as root, and you'll immediately fall into a shell without needing to input a password. You can change this behavior by using BusyBox's `passwd` command, which should have been built in to this image.

Next Steps

So, where does this leave you now? You were able to build a custom Linux distribution for the generic x86-64 architecture from open-source packages and load into it successfully. Employing the same cross-compilation toolchain, you can use a similar process to build more utilities and libraries into the operating system, such as networking utilities, storage volume management frameworks and more.

For future builds, be sure to keep the cross-compilation build directory and your headers, and be sure to continue exporting the following variables (which you probably can throw into a script file):

```
set +h
umask 022
export LJOS=~/lj-os
export LC_ALL=POSIX
export PATH=${LJOS}/cross-tools/bin:/bin:/usr/bin
unset CFLAGS
unset CXXFLAGS
export LJOS_HOST=$(echo ${MACHTYPE} | sed "s/-[^-]*/-
cross/")
export LJOS_TARGET=x86_64-unknown-linux-gnu
export LJOS_CPU=k8
export LJOS_ARCH=$(echo ${LJOS_TARGET} | sed -e 's/-
.*//'
 ↪-e 's/i.86/i386/')
export LJOS_ENDIAN=little
export CC="${LJOS_TARGET}-gcc"
export CXX="${LJOS_TARGET}-g++"
export CPP="${LJOS_TARGET}-gcc -E"
export AR="${LJOS_TARGET}-ar"
export AS="${LJOS_TARGET}-as"
export LD="${LJOS_TARGET}-ld"
export RANLIB="${LJOS_TARGET}-ranlib"
export READELF="${LJOS_TARGET}-readelf"
export STRIP="${LJOS_TARGET}-strip"
```

CONCLUSION

Linux is a modern, free operating system based on UNIX standards. It has been designed to run efficiently and reliably on common PC hardware; it also runs on a variety of other platforms, such as mobile phones. It provides a programming interface and user interface compatible with standard UNIX systems and can run a large number of UNIX applications, including an increasing number of commercially supported applications. Linux has not evolved in a vacuum. A complete Linux system includes many components that were developed independently of Linux. The core Linux operating-system kernel is entirely original, but it allows much existing free UNIX software to run, resulting in an entire UNIX-compatible operating system free from proprietary code. The Linux kernel is implemented as a traditional monolithic kernel for performance reasons, but it is modular enough in design to allow most drivers to be dynamically loaded and unloaded at run time. Linux is a multiuser system, providing protection between processes and running multiple processes according to a time-sharing scheduler. Newly created processes can share selective parts of their execution environment with their parent processes, allowing multithreaded programming. Interprocess communication is supported by both System V mechanisms—message queues, semaphores, and shared memory—and BSD's socket interface. Multiple networking protocols can be accessed simultaneously through the socket interface. The memory-management system uses page sharing and copy-on-write to minimize the duplication of data shared by different processes. Pages are loaded on demand when they are first referenced and are paged back out to backing store according to an LFU algorithm if physical memory needs to be reclaimed. To the user, the file system appears as a hierarchical directory tree that obeys UNIX semantics. Internally, Linux uses an abstraction layer to manage multiple file systems. Device-oriented, networked, and virtual file systems are supported. Device-oriented file systems access disk storage through a page cache that is unified with the virtual memory system.

REFERNCES

- https://www.educba.com/advantage-of-linux/
- https://www.elprocus.com/linux-operating-system/
- https://vissicompcodder.wordpress.com/2015/01/29/linux-design-principles-and-components-of-linux-system/
- https://www.slideshare.net/GajeshBhat1/the-linux-operating-system-a-case-study
- Book- Operating Systems
- https://www.linuxfoundation.org/projects/Linux/#:~:text=Linux%20is%20the%20world%E2%80%99s%20largest%20and%20most%20pervasive,the%20security%20and%20integrity%20of%20the%20whole%20system.
- https://www.linux.com/what-is-linux/
- https://www.guru99.com/introduction-linux.html
- https://computerbasic.net/computer-basic-guide-for-using-linux/?utm_source=ForeshopBi&utm_medium=ForeshopBi&utm_campaign=computerbasic.net&utm_term=linux%20os%20download&utm_content=Computer%20Basic%20Guide%20For%20Using%20Linux_1604366978
- https://en.wikipedia.org/wiki/List_of_Linux_distributions
- https://www.geeksforgeeks.org/introduction-to-linux-operating-system/