Assignment 4:

Write a note on Linux.

1.Introduction

Linux® is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work.

Just like Windows, iOS, and Mac OS, Linux is an operating system. In fact, one of the most popular platforms on the planet, Android, is powered by the Linux 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 (OS), the software wouldn't function.

The Linux operating system comprises several different pieces:

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.

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.

Init system – This is a sub-system that bootstraps the user space and is charged with controlling daemons. One of the most widely used init systems is systemd, which also happens to be one of the most controversial. It is the init system that manages the boot process, once the initial booting is handed over from the bootloader (i.e., GRUB or GRand Unified Bootloader).

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

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 that the users actually interact with. There are many desktop environments to choose from (GNOME, Cinnamon, Mate, Pantheon, Enlightenment, KDE, Xfce, etc.). Each desktop environment includes built-in applications (such as file managers, configuration tools, web browsers, and games).

Applications – Desktop environments do not offer the full array of apps. Just like Windows and macOS, Linux offers thousands upon thousands of high-quality software titles that can be easily found and installed. Most modern Linux distributions (more on this below) include

App Store-like tools that centralize and simplify application installation. For example, Ubuntu Linux has the Ubuntu Software Center (a rebrand of GNOME Software) which allows you to quickly search among the thousands of apps and install them from one centralized location.

2.History of Linux

- In order to understand the popularity of Linux, we need to travel back in time, about 30 years ago...
- Imagine computers as big as houses, even stadiums. While the sizes of those computers posed substantial
- problems, there was one thing that made this even worse: every computer had a different operating system.
- Software was always customized to serve a specific purpose, and software for one given system didn't run on
- another system. Being able to work with one system didn't automatically mean that you could work with
- another. It was difficult, both for the users and the system administrators.
- Computers were extremely expensive then, and sacrifices had to be made even after the original purchase just
- to get the users to understand how they worked. The total cost per unit of computing power was enormous.
- Technologically the world was not quite that advanced, so they had to live with the size for another decade. In
- 1969, a team of developers in the Bell Labs laboratories started working on a solution for the software
- problem, to address these compatibility issues. They developed a new operating system, which was
- Simple and elegant.
- Written in the C programming language instead of in assembly code.
- Able to recycle code.
- The Bell Labs developers named their project "UNIX."
- The code recycling features were very important. Until then, all commercially available computer systems
- were written in a code specifically developed for one system. UNIX on the other hand needed only a small
- piece of that special code, which is now commonly named the kernel. This kernel is the only piece of code
- that needs to be adapted for every specific system and forms the base of the UNIX system. The operating
- system and all other functions were built around this kernel and written in a higher programming language, C.

- This language was especially developed for creating the UNIX system. Using this new technique, it was much
- easier to develop an operating system that could run on many different types of hardware.
- The software vendors were quick to adapt, since they could sell ten times more software almost effortlessly.
- Weird new situations came in existence: imagine for instance computers from different vendors
- communicating in the same network, or users working on different systems without the need for extra
- education to use another computer. UNIX did a great deal to help users become compatible with different
- systems.
- Throughout the next couple of decades the development of UNIX continued. More things became possible to
- do and more hardware and software vendors added support for UNIX to their products.
- UNIX was initially found only in very large environments with mainframes and minicomputers (note that a
- PC is a "micro" computer). You had to work at a university, for the government or for large financial
- corporations in order to get your hands on a UNIX system.
- But smaller computers were being developed, and by the end of the 80's, many people had home computers.
- By that time, there were several versions of UNIX available for the PC architecture, but none of them were
- truly free and more important: they were all terribly slow, so most people ran MS DOS or Windows 3.1 on
- their home PCs.

3.Features of Linux

Open Source

- The idea behind Open Source software is rather simple: when programmers can read, distribute and change code, the code will mature. People can adapt it, fix it, debug it, and they can do it at a speed that dwarfs the performance of software developers at conventional companies. This software will be more flexible and of abetter quality than software that has been developed using the conventional channels, because more people have tested it in more different conditions than the closed software developer ever can.

- The Open Source initiative started to make this clear to the commercial world, and very slowly, commercial vendors are starting to see the point. While lots of academics and technical people have already been convinced for 20 years now that this is the way to go, commercial vendors needed applications like the Internet to make them realize they can profit from Open Source. Now Linux has grown past the stage where it was almost exclusively an academic system, useful only to a handful of people with a technical background.
- Now Linux provides more than the operating system: there is an entire infrastructure supporting the chain of effort of creating an operating system, of making and testing programs for it, of bringing everything to the users, of supplying maintenance, updates and support and customizations, etcetera. Today, Linux is ready to accept the challenge of a fast-changing world.

Linux is portable to any hardware platform:

- A vendor who wants to sell a new type of computer and who doesn't know what kind of OS his newmachine will run (say the CPU in your car or washing machine), can take a Linux kernel and make it work on his hardware, because documentation related to this activity is freely available.
- Linux was made to keep on running:
- As with UNIX, a Linux system expects to run without rebooting all the time. That is why a lot of tasks are being executed at night or scheduled automatically for other calm moments, resulting in higher availability during busier periods and a more balanced use of the hardware. This property allows for Linux to be applicable also in environments where

  people don't have the time or the possibility to control their systems night and day.

Linux is secure and versatile:

The security model used in Linux is based on the UNIX idea of security, which is known to be robust and of proven quality. But Linux is not only fit for use as a fort against enemy attacks from the Internet: it will adapt equally to other situations, utilizing the same high standards for security. Your development machine or control station will be as secure as your firewall.

Linux is scalable:

From a Palmtop with 2 MB of memory to a petabyte storage cluster with hundreds of nodes: add or remove the appropriate packages and Linux fits all. You don't need a supercomputer anymore, because you can use Linux to do big things using the building blocks provided with the system. If youwant to do little things, such as making an operating system for an embedded processor or

just recycling your old 486, Linux will do that as well.

The Linux OS and most Linux applications have very short debug-times:

Because Linux has been developed and tested by thousands of people, both errors and people to fix
them are usually found rather quickly. It sometimes happens that there are only a couple of hours
between discovery and fixing of a bug

Resource Management in Linux

- Red Hat Enterprise Linux 6 provides a new kernel feature: control groups, which are called by their Shorter name cgroups in this guide. Cgroups allow you to allocate resources—such as CPU time,System memory, network bandwidth, or combinations of these resources—among user-defined groups Of tasks (processes) running on a system. You can monitor the cgroups you configure, deny cgroups Access to certain resources, and even reconfigure your cgroups dynamically on a running system. The Cgconfig ("control group config ") service can be configured to start up at boot time and reestablish Your predefined cgroups, thus making them persistent across reboots.

- By using cgroups, system administrators gain fine-grained control over allocating, prioritizing, denying,Managing, and monitoring system resources. Hardware resources can be smartly divided up amongstTasks and users, increasing overall efficiency.

1. How Control Groups Are Organized

Cgroups are organized hierarchically, like processes, and child cgroups inherit some of the attributes

Of their parents. However, there are differences between the two models.

2. The Linux Process Model

- All processes on a Linux system are child processes of a common parent: the init process, which Is executed by the kernel at boot time and starts other processes (which may in turn start child Processes of their own).

- Because all processes descend from a single parent, the Linux process Model is a single hierarchy, or tree.

- Additionally, every Linux process except init inherits the environment (such as the PATH variable) And certain other attributes (such as open file descriptors) of its parent process.

3. The Cgroup Model

- Cgroups are similar to processes in that they are hierarchical, and ▪ child cgroups inherit certain attributes from their parent cgroup.

- The fundamental difference is that many different hierarchies of cgroups can exist imultaneously on anSystem. If the Linux process model is a single tree of processes, then the cgroup model is one or more Separate, unconnected trees of tasks (i.e. processes).

- Multiple separate hierarchies of cgroups are necessary because each hierarchy is attached to one or More subsystems. A subsystem2 Represents a single resource, such as CPU time or memory. Red Hat Enterprise Linux 6 provides nine control group subsystems, listed below by name and function.

4. Available Subsystems in Red Hat Enterprise Linux

▪ blkio — this subsystem sets limits on input/output access to and from block devices such as Physical drives (disk, solid state, USB, etc.).
Cpu — this subsystem uses the scheduler to provide cgroup tasks access to the CPU.

▪ cpuacct — this subsystem generates automatic reports on CPU resources used by tasks in a

Cgroup.

▪ cpuset — this subsystem assigns individual CPUs (on a multicore system) and memory nodes to

Tasks in a cgroup.

▪ devices — this subsystem allows or denies access to devices by tasks in a cgroup.

▪ freezer — this subsystem suspends or resumes tasks in a cgroup.

▪ memory — this subsystem sets limits on memory use by tasks in a cgroup, and generates automatic

Reports on memory resources used by those tasks.

▪ net_cls — this subsystem tags network packets with a class identifier (classid) that allows the

Linux traffic controller (tc) to identify packets originating from a particular cgroup task.

▪ ns — the namespace subsystem

5. Subsystems are also known as resource controllers

You may come across the term resource controller or simply controller in control group literature Such as the man pages or kernel documentation. Both of these terms are synonymous

with “subsystem”, and arise from the fact that a subsystem typically schedules a resource or applies a Limit to the cgroups in the hierarchy it is attached to.

The definition of a subsystem (resource controller) is quite general: it is something that acts upon A group of tasks, i.e. processes.

Relationships Between Subsystems, Hierarchies,

1. Control Groups and Tasks

Remember that system processes are called tasks in cgroup terminology. Here are a few simple rules governing the relationships between subsystems, hierarchies of cgroups,And tasks, along with explanatory consequences of those rules.

Rule 1

- Any single subsystem (such as cpu) can be attached to at most one hierarchy.
- As a consequence, the cpu subsystem can never be attached to two different hierarchies.

Rule 2

- A single hierarchy can have one or more subsystems attached to it.
- As a consequence, the cpu and memory subsystems (or any number of subsystems) can be attached To a single hierarchy, as long as each one is not attached to any other

  hierarchy.

Rule 3

- Each time a new hierarchy is created on the systems, all tasks on the system are initially members of the default cgroup of that hierarchy, which is known as the root cgroup

- For any single hierarchy you Create, each task on the system can be a member of exactly one cgroup in that hierarchy. A single Task may be in multiple cgroups, as long as each of those cgroups is in a different hierarchy. As soonAs a task is made a member of a second cgroup in the same hierarchy, it is removed from the first Cgroup in that hierarchy. At no time is a task ever in two different cgroups in the same hierarchy.

- As a consequence, if the cpu and memory subsystems are attached to a hierarchy named
- Cpu_and_mem, and the net_cls subsystem is attached to a hierarchy named net, then a running
- Httpd process could be a member of any one cgroup in cpu_and_mem, and any one cgroup in net.

- The cgroup In cpu_and_mem that the http process is a member of might restrict its CPU time to half Of that allotted to other processes, and limit its memory usage to a maximum of 1024 MB. Additionally, The cgroup in net that it is a member of might limit its transmission rate to 30 megabytes per second.

- When the first hierarchy is created, every task on the system is a member of at least one cgroup: theRoot cgroup. When using control groups, therefore, every system task is always in at least one cgroup.

Rule 4

1. Any process (task) on the system which forks itself creates a child process (task). The child task Automatically becomes members of all of the cgroups its parent is members of. The child task Can then be moved to different cgroups as needed, but initially, it always inherits the cgroups (the“environment” in process terminology) of its parent task.

2. As a consequence, consider the httpd task that is a member of the cgroup named Half_cpu_1gb_max in the cpu_and_mem hierarchy, and a member of the cgroup trans_rate_30 In the net hierarchy. When that httpd process forks itself, its child process automatically becomes aMember of the half_cpu_1gb_max cgroup, and the trans_rate_30 cgroup. It inherits the exact Same cgroups its parent task belongs to.From that point forward, the parent and child tasks are completely independent of each other:

3. Changing the cgroups that one task belongs to does not affect the other. Neither will changing groups Of a parent task affect any of its grandchildren in any way. To summarize: any child task always initially Inherit memberships to the exact same cgroups as their parent task, but those memberships can be Changed or removed later.

1. Implications for Resource Management

- Because a task can belong to only a single cgroup in any one hierarchy, there is only one way that a Task can be limited or affected by any single subsystem. This is logical: a feature, not a limitation.

- You can group several subsystems together so that they affect all tasks in a single hierarchy. Because cgroups in that hierarchy have different parameters set, those tasks will be affected Differently.

- It may sometimes be necessary to refactor a hierarchy. An example would be removing aSubsystem from a hierarchy that has several subsystems attached, and attaching it to a new,Separate hierarchy.

- Conversely, if the need for splitting subsystems among separate hierarchies is reduced, you can Remove a hierarchy and attach its subsystems to an existing one.

- The design allows for simple control group usage, such as setting a few parameters for specific Tasks in a single hierarchy, such as one with just the cpu and memory subsystems attached.

- The design also allows for highly specific configuration: each task (process) on a system could be A member of each hierarchy, each of which has a single attached subsystem. Such a configuration Would give the system administrator absolute control over all parameters for every single task.