Assignment-2 Linux

Format:Lab Session
Time:90 mins

1. In Linux FHS (Filesystem Hierarchy Standard) what is the /?

   In the Linux Filesystem Hierarchy Standard (FHS), the "/" (slash) directory is the root directory of the entire file system. All other directories, files, and devices are located within the "/" directory.

   The "/" directory is the top-level directory and contains all other directories in the system. It is the starting point for navigating the file system, and all other directories can be accessed through it.

   The "/" directory is also known as the root directory because it is the starting point for the entire file system hierarchy. Every file or directory in the file system can be traced back to the "/" directory.

   In addition to directories and files, the "/" directory also contains system-related files, such as configuration files and device files. It is also used to mount other file systems, such as network file systems or removable storage devices.

2. What is stored in each of the following paths?
   /bin, /sbin, /usr/bin and /usr/sbin
   /etc
   /home
   /var
   /tmp

/bin: The /bin directory contains essential executable files that are required to boot the system and run basic commands. This includes programs like ls, cd, cp, and other core utilities.

/sbin: The /sbin directory contains system administration executables that are used for system maintenance, network configuration, and system recovery. This includes programs like ifconfig, fdisk, and other system-level utilities.

/usr/bin: The /usr/bin directory contains the majority of user-level executable programs in Linux. This includes standard Unix utilities, as well as third-party applications that are installed on the system.

/usr/sbin: The /usr/sbin directory contains system administration executables that are used for system maintenance, network configuration, and system recovery. This includes programs like tcpdump, iptables, and other advanced system-level utilities.

/etc: The /etc directory contains configuration files for the system and its applications. This includes system-wide configuration files, startup scripts, and other important system files.

/home: The /home directory contains the home directories for all regular users on the system. Each user has their own directory under /home that contains their personal files, settings, and configurations.

/var: The /var directory contains variable data that is expected to change over time as the system runs. This includes system logs, temporary files, spool files, and other data that may change frequently.

/tmp: The /tmp directory contains temporary files that are created by the system and applications. These files are typically deleted automatically when they are no longer needed, and the directory is often cleared on system reboot.

3. What is special about the /tmp directory when compared to other directories?

   The /tmp directory is a special directory in the Linux filesystem that is used for storing temporary files. There are a few things that make the /tmp directory unique compared to other directories:

   Temporary files: The /tmp directory is specifically designated for storing temporary files. This means that any file stored in this directory is expected to be temporary and may be deleted at any time.

   File permissions: By default, the /tmp directory has very relaxed file permissions, which allows any user to create, modify, or delete files in the directory. This makes it easy for applications and users to quickly create and access temporary files.

   Automatic cleanup: Many Linux distributions are configured to automatically clean up the /tmp directory at regular intervals or on system startup. This ensures that the directory does not become cluttered with old, unused files.

   No backups: Because the /tmp directory is used for storing temporary files, it is not typically backed up. This means that any files stored in the /tmp directory should be considered disposable and should not be relied upon for long-term storage.

   In summary, the /tmp directory is special because it is designed specifically for storing temporary files, has relaxed file permissions, is automatically cleaned up, and is not typically backed up.

4. What kind of information one can find in /proc?

   The /proc directory is a virtual file system on Unix and Unix-like operating systems that provides a way for processes to communicate with the kernel. It contains a wealth of information about the current state of the system, including:

Process-related information: /proc includes subdirectories for each running process, named after the process ID. These directories contain information about the process, such as its command line arguments, environment variables, file descriptors, and memory usage.

System information: /proc includes files that provide information about system resources such as CPU usage, memory usage, disk usage, network statistics, and loaded kernel modules.

Hardware information: /proc includes files that provide information about the hardware components installed on the system, such as CPU information, interrupt information, and information about connected USB devices.

Kernel information: /proc includes files that provide information about the running kernel, such as its version number, configuration options, and parameters.

Overall, /proc is a powerful tool for system administrators and advanced users who need to monitor and diagnose system performance and behavior.

5. What makes /proc different from other filesystems?

The /proc filesystem is different from other filesystems in several ways:

It is a virtual filesystem: The /proc filesystem is not a physical filesystem that is mounted from a storage device, but rather a virtual filesystem that is created and maintained by the operating system kernel. It provides a view of the current state of the system, rather than a view of the files and directories stored on disk.

It contains runtime information: Unlike other filesystems that contain static files and directories, /proc contains runtime information about the system and the processes running on it. For example, it provides information about the currently running processes, their resource usage, and the state of the system's hardware.

It is dynamic: The information provided by /proc is dynamic and can change frequently as the system state changes. For example, the information about the currently running processes may change as new processes are started or existing processes are terminated.

It is a two-way interface: /proc provides a two-way interface between user-space processes and the kernel. User-space processes can read information from /proc to gather system information, and they can also write to /proc to modify certain system parameters.

Overall, /proc is a unique and powerful feature of Unix and Unix-like operating systems that provides a window into the current state of the system and its processes.

6. True or False? only root can create files in /proc

   False.

   In general, any user can create files in the /proc directory, although creating certain files may require root privileges. However, creating files in the /proc directory is not recommended, as it can interfere with the operation of the operating system and cause system instability. The /proc directory is intended for use by the operating system kernel and system utilities, not by end-users.

7. What can be found in /proc/cmdline?

   The /proc/cmdline file contains the command line arguments that were passed to the Linux kernel at boot time. These arguments can be used to specify various system settings and parameters, such as the system hostname, the default runlevel, and kernel parameters.

   Some examples of information that can be found in /proc/cmdline include:

   Bootloader parameters: Many Linux bootloaders, such as GRUB, allow users to specify various kernel parameters at boot time. These parameters can be used to set options such as the system language, the default timezone, and the console font.

Kernel parameters: The Linux kernel itself also accepts a wide range of parameters that can be specified at boot time. These parameters can be used to adjust kernel behavior and performance, or to enable or disable certain features.

System information: Some system information, such as the system hostname or IP address, may also be included in the kernel command line.

Overall, the /proc/cmdline file provides a convenient way to view the kernel command line arguments that were used to boot the system, which can be useful for troubleshooting and debugging system issues.

8. In which path can you find the system devices (e.g. block storage)?

   In most Unix and Unix-like operating systems, system devices such as block storage devices can be found in the /dev directory. The /dev directory contains special files that represent hardware devices and other system resources. These files are typically created by the operating system during boot time, and they can be accessed by user-space processes to interact with the underlying hardware.

   Block storage devices, such as hard drives and USB drives, are typically represented in /dev as block device files with names such as /dev/sda, /dev/sdb, etc. These files can be accessed using standard file I/O operations, and are commonly used for tasks such as mounting filesystems and copying data to and from the storage devices.

   Other types of system devices, such as network devices and input/output devices, may be represented in /dev as character device files or other types of special files, depending on the specific device and its driver.

Permissions

9. How to change the permissions of a file?

To change the permissions of a file in Unix or Unix-like systems such as Linux, you can use the chmod command. The chmod command allows you to modify the read, write, and execute permissions of the file for the file owner, group, and others.

The basic syntax of the chmod command is as follows:

bash
Copy code
```
chmod [permissions] file
```
The [permissions] argument specifies the new permissions for the file, using a combination of letters and numbers. The letters r, w, and x represent read, write, and execute permissions, respectively. The numbers 4, 2, and 1 represent read, write, and execute permissions, respectively, for the file owner, group, and others. For example, to give read and write permissions to the file owner, and read-only permissions to the group and others, you can use the following command:

bash
Copy code
```
chmod 644 file
```
In this example, 6 represents read and write permissions for the file owner (4 + 2), and 4 represents read-only permissions for the group and others (4).

You can also use the chmod command to modify the permissions of multiple files at once, using wildcard characters or by specifying a list of filenames separated by spaces. Additionally, you can use the chown and chgrp commands to change the owner and group of a file, respectively, if you have the necessary permissions to do so.

10. What does the following permissions mean?:
    777
    644
    750

The permissions for files and directories in Unix and Unix-like systems are represented by a three-digit code, where each digit represents the permissions for the file owner, group, and others, respectively. Each digit is calculated as the sum of the values of the corresponding permission bits: 4 for read, 2 for write, and 1 for execute.

Here are the meanings of the following permission codes you mentioned:

777: This means that the file or directory has read, write, and execute permissions for the file owner, group, and others. This is the highest possible level of permissions, and should be used with caution, as it can potentially allow anyone to modify or delete the file.

644: This means that the file has read and write permissions for the file owner, and read-only permissions for the group and others. This is a common set of permissions for regular files, as it allows the file owner to modify the file, while restricting access by other users.

750: This means that the file or directory has read, write, and execute permissions for the file owner, read and execute permissions for the group, and no permissions for others. This is a common set of permissions for directories that should be accessible only to a specific group of users, while restricting access by others.

11. What this command does? chmod +x some_file
    The command chmod +x some_file sets the execute permission (x) for the file some_file. The +x option adds the execute permission for the file owner, group, and others, if they do not already have it. The execute permission allows the file to be executed as a program or script, if it contains executable code.

    This command can be useful if you have a script or program that you want to run, but do not have the execute permission by default. By using chmod +x to add the execute permission, you can then run the script or program using the appropriate command, such as ./some_file for a script, or simply some_file for an executable program that is in your system's PATH.

12. Explain what is setgid and setuid
    In Unix and Unix-like operating systems, the setuid and setgid bits are special permission bits that can be set on files or directories to change the way they are executed or accessed.

    setuid: When the setuid bit is set on an executable file, the file is executed with the privileges of the file owner instead of the user who is running the file. This means that if a regular user executes a file with the setuid bit set and owned by the root user, the file will be executed with root privileges. This can be useful for certain system-level tasks that require elevated privileges, such as managing system resources or performing backups.

    setgid: When the setgid bit is set on a directory, any files or subdirectories created in that directory will inherit the group ownership of the parent directory, instead of the group ownership of the user who created them. This can be useful for managing shared resources among a group of users, such as a shared project directory.

    Note that setuid and setgid bits can also be used in combination with other permission bits, such as rwx (read, write, and execute) to create complex permission schemes for files and directories. However, it's important to use these special permission bits with caution, as they can potentially allow unauthorized access or execution of sensitive system resources.
13. What is the purpose of sticky bit?
    The sticky bit is a special permission bit that can be set on directories in Unix and Unix-like operating systems. When the sticky bit is set on a directory, only the file owner, the directory owner, or the root user can delete or rename files within that directory.

    The purpose of the sticky bit is to prevent accidental deletion or modification of important or sensitive files within a shared directory. For example, the sticky bit might be set on a directory that contains system configuration files or shared project files, to prevent a user from accidentally deleting or modifying those files.

    In addition, the sticky bit can also be used to enforce security policies for shared directories, by allowing users to create and modify their own files within the directory, but preventing them from modifying or deleting files owned by other users.

To set the sticky bit on a directory, you can use the chmod command with the +t option, followed by the name of the directory. For example, chmod +t /path/to/directory.

14. What the following commands do?
chmod
chown
Chgrp

chmod: This command is used to change the permissions of files or directories in Unix and Unix-like operating systems. It can be used to set permissions for the file owner, group, and others, for reading, writing, and executing the file. The chmod command can also be used to set special permission bits, such as the setuid, setgid, and sticky bits.

chown: This command is used to change the ownership of files or directories in Unix and Unix-like operating systems. It can be used to change the user owner and/or group owner of a file or directory. The chown command can be useful for transferring ownership of files or directories to another user or group, or for changing the ownership of system files that were accidentally modified.

chgrp: This command is used to change the group ownership of files or directories in Unix and Unix-like operating systems. It can be used to change the group owner of a file or directory to a different group. The chgrp command can be useful for managing shared resources among a group of users, or for changing the group ownership of system files that were accidentally modified.

15. What is sudo? How do you set it up?

sudo is a command that allows authorized users to run commands as another user, usually the root user, in Unix and Unix-like operating systems. This allows users to perform administrative tasks without having to log in as the root user, which can be a security risk.

To set up sudo, you can follow these steps:

Log in as the root user or a user with administrative privileges.
Install the sudo package if it is not already installed on your system.
Open the /etc/sudoers file using a text editor with administrative privileges, such as sudo visudo.
Add a line to the file that specifies the user or group that should be granted sudo privileges, and the commands or programs that they are allowed to run as the root user. For example, the line user1 ALL=(ALL) ALL grants the user user1 full sudo privileges.
Save the changes to the /etc/sudoers file and exit the text editor.
After setting up sudo, users can run commands with elevated privileges by prefixing them with the sudo command, followed by the command they want to run. For example, sudo apt-get update will run the apt-get command with elevated privileges, allowing the user to install or update system packages.

16. True or False? In order to install packages on the system one must be the root user or use the sudo command

True. In most Unix and Unix-like operating systems, installing packages or making system-level changes requires elevated privileges, which can only be granted to the root user or authorized users who can use the sudo command to run commands with root privileges.

17. Explain what are ACLs. For what use cases would you recommend to use them?

ACL stands for Access Control List, which is a set of permissions associated with a file or directory in Unix and Unix-like operating systems.

ACLs provide a finer level of control over file and directory permissions than the traditional Unix permissions model, which includes only three types of users: the file owner, the group owner, and all other users.

ACLs allow you to specify individual users and groups, along with their permissions, for a particular file or directory. This means you can grant or restrict access to specific users or groups, rather than having to rely on the broader categories of file ownership and group membership.

ACLs can be useful in a variety of use cases, including:

Multi-user systems with different levels of access requirements.
Shared directories where different users or groups need different levels of access to files and subdirectories.
Sensitive files or directories where you need to restrict access to specific users or groups.
Web servers or other applications that require fine-grained access control over files and directories.
However, it's worth noting that ACLs can be more complex to manage than traditional Unix permissions, so they may not be necessary for every use case. Additionally, not all Unix and Unix-like operating systems support ACLs, so it's important to check if your system supports ACLs before relying on them.

18. You try to create a file but it fails. Name at least three different reason as to why it could happen

There are several reasons why creating a file could fail, including:

Permission issues: The user does not have the necessary permissions to create a file in the directory or location where they are trying to create the file.

Disk space issues: There is not enough free space on the disk or partition where the file is being created.

File or directory already exists: The user is trying to create a file or directory with the same name as an existing file or directory in the same location.

File system errors: There may be file system errors or corruption that prevent the creation of new files.

Resource exhaustion: The system may have exhausted resources such as file descriptors, memory or disk I/O.

Network issues: The file is being created on a network share and there are network connectivity issues.

There may be other reasons why file creation could fail, but these are some of the most common ones.

19. A user accidentally executed the following chmod -x $(which chmod). How to fix it?

    If a user accidentally executes the command chmod -x $(which chmod), it will remove the execute permission from the chmod command, which can cause issues with managing file permissions.

    To fix this, the user with root or sudo privileges can use the chmod command to restore the execute permission on the chmod command. They can do this by running the command:

    bash
    Copy code
    ```
    sudo chmod +x $(which chmod)
    ```
    This command restores the execute permission on the chmod command, allowing it to be used to set file permissions again.

Scenarios

20. You would like to copy a file to a remote Linux host. How would you do?
    To copy a file to a remote Linux host, you can use the scp (secure copy) command. The scp command allows you to securely transfer files between hosts using the SSH protocol.

The basic syntax for using scp to copy a file from a local host to a remote host is:

ruby
Copy code
```
scp /path/to/local/file username@remote:/path/to/remote/directory
```
Where /path/to/local/file is the path to the file you want to copy, username is the username of the remote host, remote is the hostname or IP address of the remote host, and /path/to/remote/directory is the directory on the remote host where you want to copy the file.

For example, to copy a file named example.txt from your local machine to a remote host with the IP address 192.168.1.100, you could use the following command:

ruby
Copy code
```
scp /path/to/local/example.txt
user@192.168.1.100:/path/to/remote/directory
```
You will be prompted to enter the password for the remote user before the file transfer starts.

21. How to generate a random string?

In Linux, you can generate a random string using the openssl command or the uuidgen command.

To generate a random string of a specified length using the openssl command, you can use the following syntax:

perl
Copy code
```
openssl rand -hex <length>
```
Where <length> is the number of characters you want in the random string. For example, to generate a random string of 10 characters, you can run the following command:

perl
Copy code
```
openssl rand -hex 10
```
To generate a random UUID (universally unique identifier) string using the uuidgen command, you can simply run the following command:

Copy code
```
uuidgen
```
This will generate a random UUID string that is 36 characters long, consisting of numbers and letters. If you want to generate a shorter or longer random string, you can use the -r option to specify the number of random bytes to use, and then convert the output to a string using a tool like base64 or hexdump.

22. How to generate a random string of 7 characters?
    To generate a random string of 7 characters using the openssl command, you can use the following command:

    bash
    Copy code
    ```
    openssl rand -base64 9 | cut -c1-7
    ```
    This command generates a random string of 9 characters using the openssl command and the base64 encoding format. The cut command is then used to extract the first 7 characters of the output, giving you a random string of 7 characters.

    Alternatively, you can use the tr command to remove any non-alphanumeric characters from the output, like this:

    perl
    Copy code
    ```
    openssl rand -hex 4 | tr -dc 'a-zA-Z0-9' | cut -c1-7
    ```
    This command generates a random string of 4 bytes using the openssl command and the hex encoding format. The tr command is then used to

remove any non-alphanumeric characters from the output, and the cut command is used to extract the first 7 characters of the output.

Systemd

23. What is systemd?

    Systemd is a system and service manager for Linux operating systems. It is a replacement for the traditional SysVinit system initialization and service management scheme used in Unix-like systems.

    Systemd provides a range of features, such as parallel startup of system services, on-demand service activation, and support for snapshotting and restoring the system state. It also provides a central management point for various system settings and resources, such as user sessions, network interfaces, mount points, and system timers.

    Systemd has become the default system and service manager in many major Linux distributions, such as Red Hat Enterprise Linux, Fedora, Debian, and Ubuntu.

24. How to start or stop a service?

    The method to start or stop a service in Linux depends on the init system being used by the distribution. If the system is using the Systemd init system, the following commands can be used:

    To start a service:

    php
    Copy code
    ```
    sudo systemctl start <service-name>
    ```
    To stop a service:

    arduino
    Copy code
    ```
    sudo systemctl stop <service-name>
    ```

To restart a service:

php
Copy code
```
sudo systemctl restart <service-name>
```
To check the status of a service:

lua
Copy code
```
sudo systemctl status <service-name>
```
Replace <service-name> with the actual name of the service that you want to start or stop.

If the system is using another init system like SysVinit, the commands may differ slightly. You can start or stop a service using the service command, like this:

To start a service:

php
Copy code
```
sudo service <service-name> start
```
To stop a service:

arduino
Copy code
```
sudo service <service-name> stop
```
To restart a service:

php
Copy code
```
sudo service <service-name> restart
```
To check the status of a service:

lua
Copy code
```
sudo service <service-name> status
```

Again, replace <service-name> with the actual name of the service that you want to start or stop.

25. How to check the status of a service?
    To check the status of a service in Linux, you can use the systemctl command if your system is using the Systemd init system. Simply run the following command:

    lua
    Copy code
    sudo systemctl status <service-name>
    Replace <service-name> with the actual name of the service that you want to check the status of.

    If your system is using another init system like SysVinit, you can use the service command to check the status of a service. Simply run the following command:

    lua
    Copy code
    sudo service <service-name> status
    Again, replace <service-name> with the actual name of the service that you want to check the status of.

26. On a system which uses systemd, how would you display the logs?
    On a system which uses Systemd, you can display logs using the journalctl command. This command provides access to the logs collected by the systemd journal. Here are some useful examples:

    To display all logs:

    Code
    sudo journalctl
    To display logs for a specific unit/service:

    php
    Code

```
sudo journalctl -u <service-name>
```

Replace <service-name> with the actual name of the unit or service that you want to display logs for.

To display logs since the last system boot:

css
Code

```
sudo journalctl -b
```

To display logs for a specific time range:

perl
Code

```
sudo journalctl --since "2022-01-01 00:00:00" --until "2022-01-02 00:00:00"
```

Replace the date and time values with the appropriate range.

To display logs with additional information like the system hostname and process IDs:

css
Code

```
sudo journalctl -a
```

These are just a few examples of how to use journalctl command. The command provides many more options and filtering capabilities to help you quickly find the information you need.

27. Describe how to make a certain process/app a service

To make a certain process or application a service, you can create a Systemd unit file. Here are the general steps:

Create a new file in the /etc/systemd/system directory with a .service extension. For example:
bash
Copy code

```
sudo nano /etc/systemd/system/myapp.service
```

In this file, specify the necessary information about the service. Here's an example file content:

makefile
Copy code
```
[Unit]
Description=My Application
After=network.target

[Service]
ExecStart=/path/to/myapp
Restart=always
User=myuser

[Install]
WantedBy=multi-user.target
```
This file defines the service name, description, dependencies, and other important settings like the executable path, restart policy, and user account.

Save and close the file.

Reload the Systemd daemon to read the new unit file:

Copy code
```
sudo systemctl daemon-reload
```
Enable the service to start automatically on boot:
bash
Copy code
```
sudo systemctl enable myapp.service
```
Start the service:
sql
Copy code
```
sudo systemctl start myapp.service
```
Now the process or application specified in the unit file will run as a Systemd service, with all the benefits of being managed by the init system, such as automatic restarts, logging, and dependency management. You can check the status of the service using the systemctl status myapp.service command.

28. Troubleshooting and Debugging

Troubleshooting and debugging are essential skills for any Linux system administrator. Here are some general tips for troubleshooting and debugging Linux issues:

Check the logs: Most issues can be diagnosed by looking at the system logs. The logs can be found in /var/log/. Check the logs that correspond to the service or process that is having issues.

Use the command line: Many issues can be diagnosed and fixed using the command line. Use tools like ps, top, netstat, and lsof to diagnose the issue.

Check the network: If the issue involves networking, check the network configuration and connectivity. Use tools like ping, traceroute, and ifconfig to diagnose the issue.

Check the permissions: If the issue involves file or directory permissions, check the permissions using the ls -l command. Use the chmod and chown commands to modify permissions.

Check the configuration: If the issue involves configuration files, check the configuration files for syntax errors and typos. Use the diff command to compare the configuration files to a known good configuration.

Check the hardware: If the issue involves hardware, check the hardware for errors using tools like dmesg and smartctl. Check the system logs for any hardware error messages.

Check the service status: If the issue involves a service, check the service status using the systemctl status <service> command. Restart the service using the systemctl restart <service> command.

Google it: Don't be afraid to Google the error message or issue. Chances are, someone else has had the same issue and posted a solution on a forum or blog.

29. Where system logs are located?

    System logs are located in the /var/log directory in Linux systems. Each service or application may have its own log file located in this directory. Some of the commonly used log files include syslog, messages, auth.log, kernel.log, and dmesg.

30. How to follow file's content as it being appended without opening the file every time?

    You can use the tail command with the -f option to follow the content of a file as it is being appended without opening the file every time. For example, to follow the content of a file called example.log, you can use the following command:

    bash
    Copy code
    ```
    tail -f example.log
    ```
    This will display the contents of example.log and keep updating the display as new content is added to the file. To stop following the file, you can press Ctrl + C.

31. What are you using for troubleshooting and debugging network issues?

    There are several tools that can be used for troubleshooting and debugging network issues in Linux, including:

    ping: To check network connectivity and latency to a specific host.
    traceroute: To trace the route taken by packets from the source to the destination.
    netstat: To display network connections, routing tables, and other network statistics.

tcpdump: To capture and analyze network traffic in real-time.
ip: To display and manipulate routing, network devices, and network interfaces.
nslookup or dig: To perform DNS queries and resolve hostnames to IP addresses.
telnet or nc: To test network connectivity and test specific ports on a remote host.
These tools can provide valuable information for diagnosing and troubleshooting network issues in Linux.

32. What are you using for troubleshooting and debugging disk & file system issues?
There are several tools that can be used for troubleshooting and debugging disk and file system issues in Linux, including:

fsck: To check and repair the file system.
dmesg: To display kernel messages related to disk and file system errors.
lsof: To list open files and processes accessing them.
fdisk or parted: To view and manage disk partitions.
mount: To display mounted file systems and their options.
df: To display disk usage statistics for file systems.
du: To display disk usage statistics for directories and files.
These tools can help identify and fix issues with file systems and disks in Linux systems.

33. What are you using for troubleshooting and debugging process issues?
There are several tools that can be used for troubleshooting and debugging process issues in Linux, including:

ps: To display information about running processes.
top: To display real-time information about system processes, resource usage, and process statistics.

kill: To terminate a process.
pkill: To send a signal to a process based on its name.
strace: To trace system calls and signals made by a process.
lsof: To list open files and processes accessing them.
htop: An interactive version of top with more features and capabilities.
These tools can help identify and diagnose issues related to processes, such as high CPU or memory usage, crashes, and other problems that may arise.

34. What are you using for debugging CPU related issues?
For debugging CPU-related issues, there are several tools available on Linux, including:

top: To display real-time information about system processes and CPU usage.
htop: An interactive version of top with more features and capabilities.
ps: To display information about running processes, including CPU usage.
perf: A powerful performance analysis tool that can be used to profile CPU usage and identify bottlenecks.
strace: To trace system calls and signals made by a process, which can help identify any CPU-intensive operations being performed by the process.
lsof: To list open files and processes accessing them, which can help identify any processes that are using excessive CPU resources.
Using these tools, you can monitor CPU usage and identify any processes or operations that are consuming a large amount of CPU resources, helping you diagnose and resolve CPU-related issues.

35. You get a call from someone claiming "my system is SLOW". What do you do?

If I receive a call from someone claiming that their system is slow, I would follow the following steps:

Ask for more information: I would ask the user for more information about the problem they are experiencing, such as when they noticed the system slowing down, what actions they were performing at the time, and whether they have noticed any error messages or unusual behavior.

Check system resources: I would check the system's CPU, memory, and disk usage to see if there are any obvious resource constraints that could be causing the slowdown.

Check system logs: I would review the system logs to see if there are any errors or warnings that could be related to the slowdown.

Check for malware or viruses: I would run a virus scan or malware check to ensure that the system is not infected.

Check for running processes: I would check for any running processes that are consuming excessive CPU or memory resources and terminate them if necessary.

Check for disk and file system issues: I would check the disk and file system for any errors or issues that could be causing the slowdown.

Check for network issues: I would check the network connection and traffic to see if there are any issues that could be causing the slowdown.

By following these steps, I can help diagnose and resolve the issue and restore the system's performance.

36. Explain iostat output

iostat is a command-line tool that is used to monitor system input/output (I/O) device loading. The iostat output consists of a table that contains various statistics related to disk utilization and I/O activities. The table consists of the following columns:

Device: The name of the device (disk) being monitored.
tps: The number of transfers per second that were issued to the device.

kB_read/s: The amount of data in kilobytes read from the device per second.
kB_wrtn/s: The amount of data in kilobytes written to the device per second.
kB_read: The total amount of data in kilobytes read from the device.
kB_wrtn: The total amount of data in kilobytes written to the device.
%util: The percentage of time the device was busy handling I/O requests.

37. How to debug binaries?

Debugging binaries typically involves using a debugger tool, such as gdb (GNU Debugger), to analyze and modify the behavior of the binary code during runtime. Here are the basic steps to debug a binary using gdb:

Compile the binary with debugging symbols: To enable debugging, you need to compile the binary with debugging symbols. You can do this by adding the -g flag to the compiler command.

Start the binary with gdb: Run the binary with gdb using the following command: gdb <binary_file>.

Set breakpoints: Set breakpoints at specific lines or functions of the binary code that you want to analyze using the break command.

Start the program: Run the binary with the run command.

Analyze and modify the program behavior: You can use various gdb commands to analyze and modify the binary code behavior during runtime. Some useful commands are: step to execute the next line of code, next to execute the next line of code and skip over function calls, print to display the value of a variable, and set to modify the value of a variable.

Exit the debugger: When you are finished debugging, you can exit the debugger using the quit command.

38. What is the difference between CPU load and utilization?

CPU load and utilization are two different concepts that refer to different aspects of CPU performance.

CPU load is a measurement of the number of processes that are currently running on the CPU or waiting to run. It is typically expressed as a decimal number or percentage, with a value of 1.0 or 100% representing a fully utilized CPU. CPU load takes into account both running processes and processes that are waiting to run. It is a measure of the workload on the CPU, and high CPU load can indicate that the system is under heavy demand.

CPU utilization, on the other hand, is a measure of the amount of time the CPU spends executing non-idle processes. It is typically expressed as a percentage, with 100% indicating that the CPU is fully utilized. CPU utilization takes into account only running processes and not processes that are waiting to run. It is a measure of the efficiency of the CPU, and high CPU utilization can indicate that the CPU is performing well or that the system is underutilized.

In summary, CPU load measures the number of processes that are currently running or waiting to run, while CPU utilization measures the amount of time the CPU spends executing non-idle processes.

39. How you measure time execution of a program?

    There are several ways to measure the execution time of a program:

    Using the time command: Simply prefix the command with the time command, and it will print the execution time of the program along with other resource utilization statistics like CPU usage and memory consumption.

    Example: time ls -l

Using the date command: Run the command before and after the execution of the program, and calculate the difference between the two timestamps.

Example:

bash
Copy code
```
start=$(date +%s)
<command>
end=$(date +%s)
echo "Execution time: $((end - start)) seconds"
```
Using programming language-specific libraries: Many programming languages have built-in libraries or functions that can be used to measure the execution time of a program.

Example in Python:

lua
Copy code
```
import time

start = time.time()
<program>
end = time.time()

print(f"Execution time: {end - start} seconds")
```

Scenarios

40. You have a process writing to a file. You don't know which process exactly, you just know the path of the file. You would like to kill the process as it's no longer needed. How would you achieve it?

    One way to achieve this is to use the lsof command to list all processes that have the file open, and then use the kill command to terminate the process. Here are the steps:

Use the lsof command to list all processes that have the file open. For example, if the file path is /path/to/file, you can run:
bash
Copy code
lsof /path/to/file
Look for the process ID (PID) of the process that is writing to the file. It should be listed in the output of the lsof command.

Use the kill command to terminate the process. For example, if the PID of the process is 12345, you can run:

bash
Copy code
kill 12345
Note that killing a process may have unintended consequences, so it should be done with caution.

Kernel

41. What is a kernel, and what does it do?

    In computing, the kernel is a central component of an operating system (OS). It acts as a bridge between the software and the hardware, providing low-level services for memory management, process management, device management, and networking. The kernel controls all the system resources, such as the CPU, memory, and peripheral devices, and ensures that each process or task gets the necessary resources to operate correctly.
42. How do you find out which Kernel version your system is using?

To find out the kernel version your system is using, you can use the uname command with the -r option. The command uname -r will display the kernel release version, which usually consists of three numbers separated by dots. For example:

shell

Copy code
```
$ uname -r
5.11.0-41-generic
```
This output indicates that the system is using kernel version 5.11.0-41.

43. What is a Linux kernel module and how do you load a new module?

    A Linux kernel module is a piece of code that can be dynamically loaded and unloaded into the running Linux kernel. It can add new features or extend the functionality of the kernel without the need to recompile the kernel or reboot the system.

    To load a new module, you can use the insmod command followed by the name of the module file, which has the extension .ko. For example:

    ruby
    Copy code
    ```
    $ sudo insmod my_module.ko
    ```
    You can check if the module was loaded successfully by running the lsmod command. If you want to unload a module, you can use the rmmod command followed by the name of the module, like this:

    ruby
    Copy code
    ```
    $ sudo rmmod my_module
    ```
    Keep in mind that some kernel modules are essential for the system to function properly, so unloading them can cause problems or even crashes.

44. Explain user space vs. kernel space
    In operating systems, the user space and kernel space are two separate virtual address spaces. The kernel space is the part of the memory where the kernel and its drivers reside. It is protected and has complete control over the system's hardware resources. On the other hand, the user space is where the user's applications run. It is a restricted area that has no direct access to the hardware and relies on the kernel's system calls to interact with the hardware.

When a user process wants to execute a system call, it makes a request to the kernel by entering into kernel space. Once the system call is completed, the kernel returns to user space, and the user process resumes its execution.

This separation between user space and kernel space provides security and stability to the operating system. Since the kernel controls access to hardware resources, user processes cannot directly manipulate them, preventing possible system crashes or security breaches.

45. In what phases of kernel lifecycle, can you change its configuration?

    In general, you can change the kernel configuration during the build time and runtime.

    During the build time, you can modify the configuration by using the make menuconfig, make xconfig, or makeconfig command. These commands allow you to select and configure various kernel options before building the kernel.

    During runtime, you can modify the kernel configuration by editing the /proc/sys virtual file system, which contains runtime system parameters. These parameters can be changed using the sysctl command, which allows you to read, modify and set kernel parameters in real-time.

46. Where can you find kernel's configuration?

    The kernel configuration can typically be found in the /boot/config-* file or in the /proc/config.gz file if the kernel is compiled with CONFIG_IKCONFIG_PROC.

47. Where can you find the file that contains the command passed to the boot loader to run the kernel?

The file that contains the command passed to the boot loader to run the kernel depends on the specific boot loader being used.

For GRUB (Grand Unified Bootloader), the configuration file is typically located at /boot/grub/grub.cfg or /boot/grub2/grub.cfg. However, it is not recommended to modify this file directly.

Instead, changes to the boot command line can be made in the file /etc/default/grub (or /etc/default/grub2, depending on the version of GRUB being used), followed by running the command sudo update-grub to regenerate the grub.cfg file with the updated settings.

48. How to list kernel's runtime parameters?

You can list the kernel's runtime parameters using the /proc/cmdline file. This file contains the kernel command-line parameters passed by the boot loader during system startup. You can display the contents of the /proc/cmdline file using the cat command, like this:

```
bash
Copy code
cat /proc/cmdline
```
This will display a single line of text that lists all the kernel command-line parameters that were passed at boot time. Each parameter is separated by a space character.

49. Will running sysctl -a as a regular user vs. root, produce different result?

Yes, running sysctl -a as a regular user vs. root will produce different results. Many kernel parameters can only be read or modified by the root user, and attempting to access or modify these parameters as a regular user will result in a permission denied error. Running the command with root privileges will provide access to all kernel parameters.

50. You would like to enable IPv4 forwarding in the kernel, how would you do it?

    To enable IPv4 forwarding in the kernel, you need to set the value of the /proc/sys/net/ipv4/ip_forward file to 1.

    You can do this using the following command as the root user or with sudo privileges:

    bash
    Copy code
    ```
    echo 1 > /proc/sys/net/ipv4/ip_forward
    ```
    Alternatively, you can use the sysctl command to modify the kernel parameter:

    Copy code
    ```
    sysctl -w net.ipv4.ip_forward=1
    ```
    To make the change persistent across reboots, you can modify the /etc/sysctl.conf file and add the following line:

    Copy code
    ```
    net.ipv4.ip_forward = 1
    ```
    This will enable IPv4 forwarding in the kernel on the next system boot.

51. How sysctl applies the changes to kernel's runtime parameters the moment you run sysctl command?

    When you run the sysctl command to modify kernel parameters, it updates the values in the kernel's virtual file system (/proc/sys/) rather than writing them directly to the running kernel. The kernel then detects the change in this virtual file system and applies the new values to the running kernel immediately. This allows changes to be made to the kernel's runtime parameters without having to reboot the system.

52. How changes to kernel runtime parameters persist? (applied even after reboot to the system for example)

    Changes to kernel runtime parameters can be made persistent by modifying the system configuration files where these parameters are defined. The location and syntax of these files may vary depending on the specific Linux distribution and version being used, but some common examples are:

    /etc/sysctl.conf: This file contains a list of kernel parameters in the format parameter=value. Any changes made to this file will be applied at system startup.
    /etc/default/grub: This file contains the command line options passed to the kernel when booting the system. Any changes made to this file will require running update-grub command to update the bootloader configuration.
    /etc/modprobe.d/*.conf: This directory contains configuration files for kernel modules. Each file contains a list of parameters in the format options module_name parameter=value. Any changes made to these files will be applied when the corresponding module is loaded.
    After modifying these files, the changes can be applied immediately by running the sysctl -p command or rebooting the system.

53. Are the changes you make to kernel parameters in a container, affects also the kernel parameters of the host on which the container runs?

    No, the changes made to the kernel parameters in a container do not affect the kernel parameters of the host on which the container runs. This is because containers use the same kernel as the host but have their own isolated user space, which means that changes made within the container only affect the container itself and not the host.

SSH

54. What is SSH? How to check if a Linux server is running SSH?

    SSH (Secure Shell) is a network protocol used for secure remote access to a computer or server. It is typically used to log into a remote machine and execute commands, but it also supports tunneling, forwarding TCP ports, and X11 connections.

    To check if a Linux server is running SSH, you can use the following command in the terminal:

    lua
    Copy code
    ```
    systemctl status sshd
    ```
    This command will show the status of the SSH service and whether it is running or not. If the service is running, it means the server is running SSH.

55. Why SSH is considered better than telnet?

    SSH (Secure Shell) is considered better than telnet for several reasons:

    Encryption: SSH encrypts all data transmitted between the client and the server, providing secure remote access to servers and preventing eavesdropping by attackers.

    Authentication: SSH provides stronger authentication mechanisms than telnet, which relies on clear text passwords that can be easily intercepted by attackers.

    Data integrity: SSH provides data integrity checks to ensure that data is not altered during transmission.

Port forwarding: SSH allows for secure port forwarding, enabling users to securely access remote services without exposing them to the public internet.

Overall, SSH is a more secure and reliable protocol than telnet, and is widely used for remote system administration and file transfer.

56. What is stored in ~/.ssh/known_hosts?

The ~/.ssh/known_hosts file stores a list of known remote hosts for the SSH client. Each line in the file represents a host, including the hostname and the public key of the host. When the SSH client connects to a remote host, it checks the host key presented by the remote host against the keys in known_hosts file. If the key matches, the client knows that it is safe to connect to the remote host. If the key does not match, the client will display a warning that the remote host's identity is unknown or has changed.

57. You try to ssh to a server and you get "Host key verification failed". What does it mean?

The error message "Host key verification failed" indicates that the SSH client cannot verify the authenticity of the server. This can happen if the server's public key has changed, or if the client's known_hosts file has been modified.

The SSH client stores the public key of the server in the known_hosts file after the first connection to the server. On subsequent connections, the client checks the server's public key against the one stored in known_hosts. If they don't match, the client will issue the "Host key verification failed" error message to prevent a potential man-in-the-middle attack.

To resolve this issue, you can either remove the entry for the server in the known_hosts file (if you're sure that the change is legitimate), or manually verify the server's public key fingerprint and update the known_hosts file accordingly.

58. What is the difference between SSH and SSL?

    SSH and SSL are both security protocols that provide encryption and authentication, but they serve different purposes.

    SSH (Secure Shell) is a protocol used for secure remote access to a computer or server over an unsecured network. It provides secure authentication, data encryption, and confidentiality between two systems.

    SSL (Secure Sockets Layer) is a protocol used to secure communication over the internet, typically between a web server and a web browser. SSL provides encryption and authentication for data transmitted over the network and is commonly used for secure transactions, such as online shopping or banking.

    In summary, SSH is used for secure remote access to a computer or server, while SSL is used to secure communication over the internet.

59. What ssh-keygen is used for?

    The ssh-keygen command is used to generate, manage and convert authentication keys for SSH. It is typically used to generate pairs of public and private keys, which can be used for passwordless authentication to a remote server or for other purposes such as encryption or digital signatures. The ssh-keygen command can also be used to generate host keys for SSH servers. These keys are used to verify the authenticity of the server to clients when they connect.

60. What is SSH port forwarding?

    SSH port forwarding, also known as SSH tunneling, is a method of forwarding network traffic from one network port to another, over a secure SSH connection. This enables the user to access network services, such as databases, web servers, or other applications that are normally only available on a remote machine.

There are three types of SSH port forwarding:

Local port forwarding: it forwards traffic from a local port to a remote host and port, allowing the user to access a service on the remote machine as if it were local.
Remote port forwarding: it forwards traffic from a remote port to a local host and port, allowing a service running on a remote machine to be accessed from a local network.
Dynamic port forwarding: it creates a SOCKS proxy on a local port that can be used to forward traffic through an SSH connection to any destination host and port.
SSH port forwarding is a powerful and flexible feature that can be used for a variety of purposes, such as securely accessing a remote server behind a firewall, forwarding a remote desktop session over an encrypted connection, or accessing a database that is not directly accessible from the internet.