A. Access Issues

Scenario: A user reports they cannot SSH into a production server, while others can. How do you debug step by step?

B. Boot Failure

Scenario: After a reboot, a Linux server doesn't come back online. How would you debug this remotely (serial console/EC2 system log)?

C. CPU Spike

Scenario: Suddenly one service starts consuming 100% CPU and slows down the entire system. Walk me through your investigation.

D. Disk Full

Scenario: Your app crashes due to `No space left on device`, but `df -h` shows free space. How do you resolve this?

E. Exit Code Failures

Scenario: A cronjob fails with exit code `137`. What does it mean, and how do you debug it?

F. File Descriptor Leak

Scenario: A Java service suddenly stops accepting new connections. Logs show `Too many open files`. How do you trace and fix the leak?

G. Graceful Shutdown

Scenario: A service doesn't stop cleanly after `systemctl stop`. How would you troubleshoot zombie/defunct processes?

H. High Load Average

Scenario: Load average is 30, but CPU utilization is only 20%. What's happening and how do you prove it?

I. Inode Exhaustion

Scenario: Disk has space, but app still cannot write files. How do you debug inode exhaustion?

J. Journalctl Debugging

Scenario: A systemd service starts but immediately exits. How do you use journalctl to find the root cause?

K. Kernel Panic

Scenario: Your VM restarts automatically and system logs show `kernel panic`. How do you capture details for RCA?

L. Latency Investigation

Scenario: Users report high latency connecting to your API server. How do you check if it's network latency, DNS, or app issue?

M. Memory Leak / OOM

Scenario: A service is killed by OOM killer. How do you confirm which process caused it, and how would you prevent it?

N. Network Troubleshooting

Scenario: You can `ping` a server, but cannot `curl` a service running on port 8080. Walk through the debug steps.

O. Orphan Processes

Scenario: After a service restart, you see multiple orphan processes running. How do you handle them and prevent reoccurrence?

P. Permission Denied

Scenario: A deployment script fails with `Permission denied`, but the file is `chmod +x`. How do you debug SELinux/AppArmor related issues?

Q. Quota Issues

Scenario: A user cannot create new files even though disk is free. You suspect user quota limits. How do you verify and fix?

R. Read-Only Filesystem

Scenario: A containerized app crashes with `Read-only file system`. How do you debug mount options and fix?

S. SSH Latency

Scenario: SSH login to a server takes 2 minutes before showing a prompt. What could cause this and how do you fix it?

T. TCP Troubleshooting

Scenario: A web app is timing out. How do you verify the 3-way TCP handshake (SYN, SYN-ACK, ACK) and locate failure?

U. Ulimit Issues

Scenario: A Node.js app crashes under load. Logs show `EMFILE: too many open files`. How do you debug and fix?

V. Volume Mount Issues

Scenario: An NFS mount inside a Linux server hangs and blocks I/O. How do you debug and safely unmount?

W. Wait States

Scenario: Processes are stuck in `D` (uninterruptible sleep). How do you debug what they're waiting on (disk, NFS, kernel locks)?

X. XFS/EXT4 Differences

Scenario: Your database team asks why EXT4 volumes behave differently from XFS under high parallel writes. How do you explain?

Y. YAML Misconfig

Scenario: A Kubernetes pod won't start, logs are fine, but `kubectl describe` shows `invalid mount path`. How do you catch YAML syntax errors in Linux?

Z. Zombie Processes

Scenario: Monitoring alerts show dozens of zombies (`Z` state). How do you confirm the parent process and clean them up?

Answers:

A — Access Issues (SSH)

Symptom: One user cannot SSH to prod while others can.

Verify

```
# From user machine

ssh -vvv user@host

# On host (from admin)

sudo tail -n200 /var/log/auth.log         # or /var/log/secure

ss -tnp | grep :22
```

Debug

- Check client `~/.ssh/` keys, `ssh -vvv` output for auth step.
- On server: `ss`/`netstat` to see if sshd listening.
- Check `/etc/ssh/sshd_config` for `AllowUsers`, `AllowGroups`, `DenyUsers`.

Fix

- If key issue: re-add public key to `~/.ssh/authorized_keys`.
- If blocked: remove ban from fail2ban or update firewall.
- Restart sshd if config changed: `sudo systemctl restart sshd`.

Prevent

- Centralized auth (SSO/LDAP), ephemeral access (bastion + session recording).

Interview Point: mention logs and `ssh -vvv` first, then policies (AllowUsers) and network/iptables.

B — Boot Failure

Symptom: Server doesn't come up after reboot.

Verify

- For cloud (EC2/GCP): check instance system log / serial console in cloud console.
- For physical/VM: access serial/console.

Debug

```
# From serial console logs — look for kernel panics, fsck errors

journalctl -b -1 -k

# If filesystem errors, look for fsck output in console
```

Fix

- If kernel panic due to bad kernel: boot into older kernel via grub.
- If fs corruption: boot rescue, `fsck -y /dev/sdX`.
- If misconfigured fstab: edit fstab in initramfs or rescue and comment offending mount.

Prevent

- Validate fstab entries, test kernel upgrades in staging, automatic fsck scheduling, automated boot health checks.

Interview Point: remote rescue strategies (cloud serial console, initramfs shell, AMI rollback).

C — CPU Spike

Symptom: One service consumes 100% CPU.

Verify

```
top -b -o %CPU | head
```

```
pidstat -u 1
```

```
ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head
```

Debug

- `strace -p <pid>` to see syscalls.
- `perf top` / `perf record` for hotspots (if permitted).
- Check logs for infinite loops or spin-wait.

Fix

- Kill or throttle process: `kill <pid>` or `renice +10 <pid>`; fix code or restart service.
- Add CPU limits (systemd `CPUShares`, cgroups, or container limits).

Prevent

- Resource limits, autoscaling, profiled code paths, circuit breakers.

Interview Point: mention difference between CPU-bound and spin-wait; use `perf` for kernel-level hotspots.

D — Disk Full but df shows free

Symptom: App crashes `No space`, but `df -h` shows space.

Verify

```
df -h
```

```
df -i

lsof +L1        # files deleted but held open
```

Debug

- `lsof +L1` finds deleted files still held by processes.
- Check inodes: `df -i`.

Fix

- Restart or kill the process holding deleted files to free space.

```
lsof | grep deleted

kill -HUP <pid>   # gracefully close or restart process
```

- If inode exhaustion: remove many small files or recreate filesystem with more inodes.

Prevent

- Log rotation, `logrotate`, avoid writing huge temp files to root, monitoring disk/inode alerts.

Interview Point: explain deleted-but-open file behavior and inode exhaustion difference.

E — Exit Code 137 (OOM / SIGKILL)

Symptom: Cronjob or container exits with code 137.

Verify

```
# 137 = 128 + 9 => SIGKILL

dmesg | grep -i -E 'oom|killed'
```

```
journalctl -u <service> --since "1 hour ago"
```

Debug

- Check OOM killer logs in `dmesg`.
- Identify if process exceeded cgroup/container memory limits.

Fix

- Increase memory quota, add swap (carefully), or fix memory leak.
- For containers: adjust Kubernetes `resources.requests/limits`.

Prevent

- Memory limits, monitoring, heap dumps and profiling.

Interview Point: translate exit codes, search OOM logs in kernel messages.

F — File Descriptor Leak

Symptom: Service `Too many open files`.

Verify

```
ulimit -n

lsof -p <pid> | wc -l

lsof -p <pid> | head
```

Debug

- Use `lsof` to list FDs and type (sockets/files).

- `strace -p <pid> -e trace=desc` to see open/close syscalls.

Fix

- Fix code to close FDs; temporary: restart service.
- Increase FD limits: `/etc/security/limits.conf` and systemd `LimitNOFILE=`.

Prevent

- Proper resource lifetime management, alerts on FD usage.

Interview Point: show `strace` to prove leak; discuss ephemeral vs long-lived connections.

G — Graceful Shutdown / Defunct Processes

Symptom: `systemctl stop` doesn't stop service; zombie/defunct left.

Verify

```
ps -el | grep Z
```

```
ps -o pid,ppid,stat,cmd -p <zombie-pid>
```

Debug

- Zombies (`Z`) are waiting for parent to reap them. Identify parent (`ppid`).

Fix

- Restart or kill the parent process; `kill -SIGCHLD <parent>` sometimes helps.
- If parent is init-like, reboot as last resort.

Prevent

- Ensure parent process reaps children (use `wait()` or proper process management), use systemd service with `KillMode=control-group`.

Interview Point: explain difference between defunct (zombie) and orphan; `init` adoption.

H — High Load Average, Low CPU Util

Symptom: Load avg = 30, CPU usage = 20%.

Verify

```
uptime

top

iostat -xz 1

vmstat 1

ps -eo pid,stat,cmd | grep ' D'   # D = uninterruptible sleep
```

Debug

- High load with low CPU usually means I/O wait or tasks blocked (D state).
- `iostat` shows device utilization and await times.

Fix

- Identify slow device, check underlying storage, fix kernel-level issues or offload IO.
- If NFS, check server. If disk, check SMART, replace disk.

Prevent

- Disk capacity planning, faster storage, limits on sync operations.

Interview Point: show `ps` D-state processes and use `iostat` to correlate.

I — Inode Exhaustion

Symptom: Disk has space but cannot create files.

Verify

```
df -h
```

```
df -i
```

```
find /path -xdev -printf '%h\n' | sort | uniq -c | sort -nr | head
```

Debug

- `df -i` to show inode usage.
- Find directories with many small files.

Fix

- Delete unnecessary small files, archive them, recreate filesystem with different inode ratio (reformat), or use XFS.

Prevent

- Avoid many tiny files; use object stores; implement cleanup policies.

Interview Point: difference between block usage and inode limits.

J — Journalctl Debugging (systemd services)

Symptom: systemd service starts then exits.

Verify

```
systemctl status mysvc
```

```
journalctl -u mysvc -b --no-pager

journalctl -xe
```

Debug

- Inspect journal for error message, stacktrace, dependency failures.
- Check `ExecStart` in unit, `Restart=` policy.

Fix

- Fix executable path/permissions or environment variables, reload unit files, `systemctl daemon-reload`, `systemctl restart mysvc`.

Prevent

- Unit test service files, define `RestartSec` and health checks.

Interview Point: use timestamps and `-o cat` for clean logs; explain unit dependencies.

K — Kernel Panic

Symptom: VM restarts; logs show `kernel panic`.

Verify

- Pull serial console logs or `/var/log/kern.log` if available.
- For cloud, check instance console output.

Debug

- Look for panic stacktrace, module names, recent kernel updates or OOM killers.
- If reproducible: boot with `panic=0` disabled to keep console.

Fix

- Revert faulty kernel modules, upgrade kernel or drivers, check hardware.

Prevent

- Test kernel updates in staging, use livepatch where possible.

Interview Point: capturing panic info from serial console is critical for RCA.

L — Latency Investigation

Symptom: API high latency for users.

Verify

```
# Network

ping -c 10 api.example.com

traceroute api.example.com

# App

curl -w '%{time_total}\n' -o /dev/null -s http://api/endpoint

# DB

EXPLAIN ANALYZE <query>
```

Debug

- Use distributed tracing (OpenTelemetry/Jaeger) to find slow hop.
- Compare network RTT vs app processing time vs DB time.

Fix

- Tune slow DB queries, add caches, reduce serialization, scale horizontally.

Prevent

- SLO-based alerting, key-path tracing, profiling.

Interview Point: emphasize span-level tracing to isolate network vs app vs DB.

M — Memory Leak / OOM

Symptom: Service killed by OOM.

Verify

```
dmesg | grep -i oom

ps aux --sort=-%mem | head

# For Java

jcmd <pid> GC.heap_info

jmap -heap <pid>
```

Debug

- Identify culprit with `ps` and `top`.
- Take heap dump (Java: `jmap -dump`) and analyze with MAT.

Fix

- Fix application memory leak, increase memory, enable cgroup limits, restart service as interim.

Prevent

- Memory limits, observability (heap metrics), automated restarts, memory leak unit tests.

Interview Point: show OOM logs and explain process selection by OOM killer.

N — Network: ping works, curl fails

Symptom: Can ping host but cannot curl service port.

Verify

```
ss -tulnp | grep 8080

curl -v host:8080

telnet host 8080

iptables -L -n

sudo nft list ruleset      # if nftables in use
```

Debug

- Check if service listening on 127.0.0.1 vs 0.0.0.0.
- Check firewall (iptables, nftables, security groups).
- Use `tcpdump -i any port 8080` to see incoming packets.

Fix

- Bind service to correct interface, update firewall rules or security group.

Prevent

- Standardize host binding, e2e connectivity tests, health checks.

Interview Point: differentiate network reachability (ICMP) vs TCP service availability.

O — Orphan Processes after restart

Symptom: Multiple old processes remain after service restart.

Verify

```
ps aux | grep myservice

pstree -p <parent-pid>

systemctl status myservice
```

Debug

- Parent might spawn children outside cgroup; check process tree.
- systemd `KillMode` default is `control-group` — ensure service created with `Type=forking` handled properly.

Fix

- Use systemd to manage cgroups properly; set `KillMode=control-group`.
- Kill orphan processes and fix service to not background children incorrectly.

Prevent

- Proper process model with systemd, avoid daemonizing inside container.

Interview Point: discuss supervision trees and process control groups.

P — Permission Denied (SELinux/AppArmor)

Symptom: `Permission denied` even when perms look correct.

Verify

```
# AppArmor

sudo aa-status

# SELinux

getenforce

ausearch -m avc -ts recent

# Check audit logs

sudo ausearch -m avc -ts recent | aureport -f
```

Debug

- SELinux/AppArmor can block access despite 755 perms.
- Check audit logs for denials.

Fix

- Adjust SELinux context: `chcon -t httpd_sys_content_t /var/www/html -R` or create policy.
- For AppArmor: add profile rule or put in complain mode.

Prevent

- Document and include SELinux/AppArmor in CI tests.

Interview Point: differentiate Unix perms vs MAC policies and audit-first approach.

Q — Quota Issues

Symptom: User cannot create files—quota suspected.

Verify

```
repquota -a

quota -u username

edquota -u username
```

Debug

- Check user or group quota, filesystem must be mounted with quota options.

Fix

- Increase quota with `edquota` or clear files, or migrate to larger FS.

Prevent

- Monitor quota usage, warn users, automate cleanup.

Interview Point: quotas often overlooked; show `repquota` output.

R — Read-Only Filesystem

Symptom: `Read-only file system` errors.

Verify

```
mount | grep 'ro,'

dmesg | tail -n50

cat /proc/mounts
```

Debug

- Kernel may remount FS read-only on error. Check `dmesg` for I/O errors.

- Check mount options, NFS server state if NFS.

Fix

- Remount read-write if safe: `mount -o remount,rw /mountpoint` (only if hardware OK).
- If corruption, run fsck in maintenance mode.

Prevent

- Replace failing disks, robust mounts, monitor disk health.

Interview Point: don't remount RW without understanding cause; look at kernel logs first.

S — SSH Login Slow

Symptom: SSH prompt takes long after password accepted.

Verify

```
ssh -vvv host

# On host:

sshd -T | grep UseDNS

# Check DNS reverse lookups:

getent hosts <client-ip>
```

Debug

- Slow DNS reverse lookup (`UseDNS yes`), GSSAPI auth or home dir NFS slow.
- Check auth logs, `sshd_config` `UseDNS` and `GSSAPIAuthentication`.

Fix

- Disable `UseDNS` in `/etc/ssh/sshd_config`.
- Disable GSSAPI if unused. Ensure home dirs responsive.

Prevent

- Fast DNS, avoid blocking calls in login flow.

Interview Point: common cause is DNS/ident lookup; `ssh -vvv` helps locate stage.

T — TCP Handshake Failures

Symptom: Clients time out; SYN not progressing.

Verify

```
# Server side

tcpdump -n -i any 'tcp[tcpflags] & tcp-syn != 0 and port 80'

ss -s

iptables -L -n
```

Debug

- Use `tcpdump` to see SYN, SYN-ACK behavior.
- If no SYN arrives: routing/firewall issue; SYN arrives but no SYN-ACK: server not listening or blackholed.

Fix

- Fix firewall/security groups, ensure server listening, correct routing.

Prevent

- Network tests, health checks, iptables rules in IaC.

Interview Point: describe reading `tcpdump` captures to identify missing packet in handshake.

U — ulimit / too many open files

Symptom: Node.js or other app `EMFILE`.

Verify

```
ulimit -n

cat /proc/<pid>/limits

lsof -p <pid> | wc -l
```

Debug

- Increase per-process limits: edit `/etc/security/limits.conf`.
- For systemd services: set `LimitNOFILE=` in unit file and `systemctl daemon-reload`.

Fix

- Tune ulimit and fix leaks; restart service.

Prevent

- Set reasonable defaults, monitor file descriptor consumption.

Interview Point: show `cat /proc/<pid>/limits` to demonstrate current limits.

V — Volume / NFS Mount Hangs

Symptom: NFS mount hangs, `ls` blocks.

Verify

```
mount | grep nfs

showmount -e nfshost

# On client observe blocked processes:

ps -eo pid,stat,cmd | grep '^ *[0-9]* D'
```

Debug

- Kernel shows processes in D state waiting on NFS.
- `strace` will hang too; use `umount -f` or `umount -l` (lazy) to detach.
- Check NFS server health and network.

Fix

- If server down, bring it back or use `umount -l /mnt` then remount.
- Kill blocked processes after remount.

Prevent

- Use timeouts, mount options (soft vs hard), monitoring for NFS server.

Interview Point: emphasize careful use of forced/lazy unmounts to avoid data loss.

W — Wait States (D)

Symptom: Many processes in `D` (uninterruptible sleep).

Verify

```
ps axo pid,stat,cmd | egrep ' D '
```

```
iostat -xz 1

dmesg | tail
```

Debug

- D-state usually means waiting on I/O (disk/NFS/kernel locks).
- Correlate with `iostat` or storage controller errors.

Fix

- Fix underlying IO subsystem (replace faulty disk, fix network storage).
- Reboot may be required if processes stuck and device unusable.

Prevent

- Monitor I/O latency, avoid blocking operations without timeouts.

Interview Point: D-states are kernel-level; userland cannot kill them until IO returns.

X — XFS vs EXT4 behavior under load

Scenario Answer

- **XFS** scales better under high parallel writes, metadata journaling optimized for large files and high concurrency.
- **EXT4** can be faster for small-file workloads and simpler workloads.
- For DBs with many small random writes, test both in your environment; XFS often preferred for parallel I/O (cloud recommended XFS for CSI volumes).

Interview Point: recommend benchmarking (fio) rather than blanket statements.

Y — YAML misconfig (Kubernetes)

Symptom: Pod `Pending` or CrashLoopBackOff due to manifest error.

Verify

```
kubectl apply -f pod.yaml --dry-run=server -o yaml

kubectl describe pod <pod>

kubectl logs <pod> --previous
```

Debug

- `kubectl describe` shows events like `MountVolume.SetUp failed` or `ErrImagePull`.
- Validate YAML spacing/indentation; missing `resources` or wrong `apiVersion`.

Fix

- Correct YAML, `kubectl apply -f`, check `kubectl get events`.

Prevent

- Use `kubectl apply --validate` or CI lint (kubeval, yamllint).

Interview Point: use server-side dry run and `kubectl explain` for schema.

Z — Zombie Processes

Symptom: Many `Z` processes visible.

Verify

```
ps -eo pid,ppid,stat,cmd | awk '$3 ~ /Z/ {print}'

# or
```

```
ps -el | grep Z
```

Debug

- For each zombie, find PPID. The parent has to `wait()` on child; if parent hung, zombies accumulate.

Fix

- Restart or kill parent process; if parent is PID 1, reboot.
- `kill -SIGCHLD <parent-pid>` may help if parent handles SIGCHLD.

Prevent

- Ensure parent processes reap children, use process supervisors, adopt safe fork/exec patterns.

Interview Point: explain resource impact (zombies use pid table entries only) and correct mitigation.