Low Latency Linux Administrator

Technical Interview Preparation Guide

SECTION 1: TECHNICAL INTERVIEW QUESTIONS & SAMPLE ANSWERS

1. What is kernel bypass and why is it critical in low-latency trading environments?

In Simple Terms:

Normally, when your computer receives data from the network, the operating system (OS kernel) acts like a middleman, passing data to your application. This middleman adds tiny delays. Kernel bypass removes this middleman so data goes directly from the network card to your application — much faster.

Technical Context:

Kernel bypass (e.g., DPDK, OpenOnload/Solarflare, RDMA) allows network data to transfer directly between NIC and user-space application memory, bypassing the OS kernel's TCP/IP stack and context-switch overhead. This eliminates interrupt handling, syscall overhead, and memory copies, reducing latency from microseconds to nanoseconds.

Sample Answer:

In a trading environment, kernel bypass frameworks like DPDK or OpenOnload (Solarflare) are used to reduce network latency to sub-microsecond levels. The NIC is assigned directly to user-space via VFIO/UIO drivers, memory is mapped using hugepages, and the application polls the NIC ring buffers directly (busy-polling) instead of relying on interrupts. This is critical for HFT where microseconds matter.

2. How do you tune a Linux system for low latency? Walk me through your approach.

In Simple Terms:

Think of a road with traffic lights — in normal mode, cars stop and go constantly. For low latency, we want a highway with no traffic lights. We tune the system to keep our important processes running without interruptions.

Technical Context:

Low-latency Linux tuning involves CPU isolation, interrupt affinity, NUMA awareness, hugepages, tickless kernel, and disabling power management features that introduce jitter.

Sample Answer:

My approach: (1) CPU isolation: isolcpus, rcu_nocbs, nohz_full kernel params. (2) IRQ affinity: assign NIC interrupts to specific CPUs away from application CPUs. (3) NUMA: pin processes to NUMA node local to NIC using numactl. (4) Hugepages: allocate 1GB/2MB pages for DPDK. (5) Disable C-states/P-states: processor.max_cstate=1, intel_idle.max_cstate=0. (6) Disable hyperthreading for latency-sensitive cores. (7) Set CPU governor to 'performance'. (8) Disable NUMA balancing: kernel.numa_balancing=0. (9) Tune NIC ring buffers and disable interrupt coalescing.

3. A production trading system is experiencing intermittent latency spikes. How do you diagnose?

In Simple Terms:

Imagine a sports car that usually runs perfectly but randomly stutters for a split second. We need to catch that stutter happening and find out what's causing it — is it the engine, the fuel, or road bumps?

Technical Context:

Latency spikes can be caused by OS jitter (scheduling, interrupts), GC pauses, memory allocation, NIC issues, kernel daemons, or thermal throttling.

Sample Answer:

Step-by-step: (1) Use 'cyclictest' to measure OS jitter baseline. (2) Check for SMI interrupts: read MSR 0x34 before/after spike. (3) Review /proc/interrupts for unexpected interrupt storms. (4) Check CPU frequency throttling: turbostat or /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq. (5) Analyze NIC stats: ethtool -S ethX for dropped packets. (6) Check kernel logs: dmesg -T for thermal throttle events. (7) Use 'perf stat' to look for cache misses or branch mispredictions. (8) Review /proc/softirqs for soft interrupt distribution.

4. Explain NUMA and how it impacts performance in distributed trading systems.

In Simple Terms:

Your computer has multiple processors, each with its own nearby memory (like a desk with its own drawer). If one processor needs to use the other processor's memory, it takes longer — like reaching across the room for files. NUMA-aware systems keep everything nearby.

Technical Context:

Non-Uniform Memory Access (NUMA) architecture has multiple nodes, each with CPUs and local memory. Cross-node memory access has higher latency (100-300ns extra). Proper NUMA binding is critical for latency-sensitive workloads.

Sample Answer:

Commands to diagnose: 'numactl --hardware' shows topology, 'numastat' shows per-node memory access stats. For binding: 'numactl --cpunodebind=0 --membind=0 ./app'. In kernel config, check

/proc/sys/kernel/numa_balancing (should be 0 for trading). Also verify NIC NUMA locality: cat /sys/class/net/ethX/device/numa_node should match the NUMA node of your application CPUs.

5. How do you manage hugepages for DPDK-based applications?

In Simple Terms:

Computers organize memory in small pages, like filing documents in tiny folders. Hugepages use big folders — this means less time spent finding and organizing files, which is faster for high-speed applications.

Technical Context:

Standard page size is 4KB. Hugepages are 2MB or 1GB. Fewer TLB entries needed = fewer TLB misses = lower latency. DPDK requires hugepages for its memory pool.

Sample Answer:

Configuration: echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages for 2MB pages. For 1GB pages: add 'hugepagesz=1G hugepages=4' to kernel cmdline. Mount hugetlbfs: 'mount -t hugetlbfs nodev /mnt/huge'. Verify: 'cat /proc/meminfo | grep Huge'. For persistent config: add to /etc/sysctl.conf: vm.nr_hugepages=1024. DPDK then maps these via --socket-mem parameter.

6. What is your experience with Red Hat Linux in enterprise environments? How do you manage updates in production?

In Simple Terms:

Red Hat Linux is like a car brand known for reliability and long-term support — enterprise companies trust it. Managing updates in production is like servicing a car fleet: you can't take all cars off the road at once.

Technical Context:

RHEL uses RPM packaging, subscription-manager for licensing, and supports Satellite/Foreman for centralized patch management. Rolling updates with maintenance windows are standard practice.

Sample Answer:

I use subscription-manager to manage entitlements, 'yum/dnf' for package management. In production: (1) Test updates in staging first. (2) Use 'yum --downloadonly' to pre-stage packages. (3) Apply during maintenance windows using automation (Ansible). (4) Use 'needs-restarting -r' to check if reboot required. (5) Satellite Server for centralized content views. (6) RPM version pinning with yum versionlock. (7) rollback capability via 'yum history undo'.

7. How do you implement and manage configuration management at scale?

In Simple Terms:

When you have thousands of servers, you can't configure each one manually. Configuration management tools are like a master recipe book — you define what every server should look like, and the tool automatically makes it so.

Technical Context:

Tools like Ansible, Puppet, Chef, or Salt enforce idempotent configurations across server fleets. Integration with CI/CD ensures configs are version-controlled and tested before deployment.

Sample Answer:

I use Ansible for configuration management due to its agentless architecture. Key practices: (1) Store all playbooks in Git with branching strategy. (2) Use Ansible Tower/AWX for RBAC and audit trails. (3) Idempotent playbooks — always use 'ansible-lint' before merge. (4) Ansible Vault for secrets management. (5) Dynamic inventory from CMDB/DNS. (6) Regular 'ansible-playbook --check' (dry-run) before production apply. (7) Use roles for reusable components. (8) Integration with Jenkins/GitLab CI for automated testing.

8. Describe your experience with DNS, DHCP, and NTP in large-scale environments.

In Simple Terms:

DNS is like a phone book (converts names to addresses), DHCP automatically hands out IP addresses to new devices like a hotel giving room keys, and NTP keeps all clocks synchronized — critical in trading where timestamps matter to microseconds.

Technical Context:

In trading environments, NTP accuracy is critical. PTP (IEEE 1588) achieves sub-microsecond synchronization. DNS must be highly available with proper caching. DHCP should have failover pairs.

Sample Answer:

DNS: BIND or Unbound, split-horizon for internal/external, DNSSEC where required. Monitor with 'dig', 'named-checkconf'. DHCP: ISC DHCP with failover pairs, reservations for servers, IPAM integration. NTP: In trading, use PTP with hardware timestamping for sub-microsecond accuracy. Chrony for software sync (chronyc tracking, chronyc sources -v). Verify time sync: 'timedatectl', monitor offset in /var/log/chrony.

9. How do you approach change management in a 24/7 production trading environment?

In Simple Terms:

A trading system runs all day, every day. Making changes is like repairing an airplane engine while it's flying — it requires strict planning, approvals, rollback plans, and communication.

Technical Context:

ITIL-based change management with CAB (Change Advisory Board) approval, defined maintenance windows, rollback procedures, and post-implementation review.

Sample Answer:

Process: (1) RFC (Request for Change) submitted with impact analysis, risk assessment, rollback plan. (2) CAB review for standard/normal changes. (3) Pre-change: notify stakeholders, prepare rollback scripts, take system snapshots. (4) Freeze periods during market hours (typically 09:00-16:00 EST). (5) Emergency changes: expedited approval with senior sign-off. (6) Post-change: monitoring period, performance baseline comparison. (7) All changes tracked in ServiceNow/Jira with full audit trail.

10. How do you monitor production Linux infrastructure? What tools and metrics do you focus on?

In Simple Terms:

Monitoring is like a dashboard in a car — it shows you speed, fuel, temperature so you know if something is wrong before it breaks down. For servers, we monitor CPU, memory, disk, and network.

Technical Context:

Production monitoring requires real-time metrics, alerting, log aggregation, and distributed tracing. Key metrics: CPU utilization, memory pressure, I/O wait, network saturation, and application-specific latency percentiles.

Sample Answer:

Stack: Prometheus + Grafana for metrics, ELK stack for logs, PagerDuty for alerting. Key metrics I monitor: CPU steal time (for VMs), iowait (storage bottleneck), network retransmits (ethtool -S), memory reclaim activity (sar -r), process context switches (pidstat -w), and application-specific p99/p999 latencies. For trading: NIC hardware timestamps, kernel interrupt latency via cyclictest, and SMI count via MSR reads.

11. Explain TCP tuning for low-latency financial applications.

In Simple Terms:

TCP is the protocol computers use to communicate reliably. By default it's tuned for general use. For trading, we tune it like a race car — strip out anything that adds delay.

Technical Context:

TCP tuning involves buffer sizes, congestion control, Nagle algorithm, and socket options specific to latency-sensitive workloads.

Sample Answer:

Key tunings: (1) Disable Nagle: TCP_NODELAY socket option or sysctl net.ipv4.tcp_nodelay. (2) Increase socket buffers: net.core.rmem_max, wmem_max. (3) Use TCP_QUICKACK. (4) Enable TCP_TIMESTAMPS=0 if not needed (reduces overhead). (5) net.ipv4.tcp_congestion_control=bbr or

cubic. (6) Increase backlog: net.core.somaxconn, net.ipv4.tcp_max_syn_backlog. (7) net.ipv4.tcp_syn_retries=2 for faster failure detection. (8) Consider UDP with custom reliability layer for ultra-low latency multicast market data.

12. How do you handle disk I/O performance issues in production?

In Simple Terms:

If your hard drive is slow, the whole computer slows down — like a restaurant kitchen where the food prep station is a bottleneck. We diagnose which disk, which process, and why it's slow.

Technical Context:

I/O performance issues can be caused by queue depth saturation, filesystem fragmentation, incorrect I/O scheduler, RAID misconfiguration, or competing workloads.

Sample Answer:

Diagnosis: (1) iostat -x 1 — check %util, await, svctm. (2) iotop — identify top I/O consumers. (3) blktrace/blkparse — detailed I/O trace. (4) lsblk -t — check I/O scheduler (use 'none' or 'mq-deadline' for SSDs/NVMe). (5) Check filesystem: xfs_info, tune2fs -l. (6) For NVMe: nvme-cli stats. Resolution: Change I/O scheduler, increase queue depth (echo 64 > /sys/block/nvmeXnX/queue/nr_requests), use noatime mount option, consider RAM-backed tmpfs for temp data.

13. What is your experience with Automated Lights Out Management (ALOM/IPMI/iDRAC/iLO)?

In Simple Terms:

Out-of-band management is like being able to remotely restart a server even when it's completely crashed — like a remote control that works even when the TV is off.

Technical Context:

IPMI, iDRAC (Dell), iLO (HP), and BMC-based management allow out-of-band access for power control, console access, hardware monitoring, and firmware updates without requiring OS.

Sample Answer:

Tools: ipmitool for IPMI, racadm for iDRAC, hponcfg/ssacli for iLO. Common operations: 'ipmitool power status/on/off/reset', 'ipmitool sol activate' for serial-over-LAN console, 'ipmitool sensor list' for hardware health. I configure dedicated IPMI VLANs, strong authentication, and integrate with Ansible for automated BMC configuration. Critical for bare-metal provisioning workflows and after-hours incident response.

14. How do you build systems from scratch in a large enterprise? Describe your automated provisioning workflow.

In Simple Terms:

Building servers manually is slow and error-prone — like cooking each meal from scratch every time. Automated provisioning is like having a recipe that a robot can follow perfectly every time.

Technical Context:

Modern bare-metal provisioning uses PXE boot, Kickstart/Preseed, and post-install configuration management. Cloud-like workflows for physical hardware.

Sample Answer:

Workflow: (1) Network boot via PXE (BIOS/UEFI). (2) DHCP assigns IP and points to TFTP server. (3) Kickstart file provides automated RHEL installation with partitioning, network, users. (4) Post-install: register with Satellite/RHN, apply base Ansible playbooks. (5) Hardware configuration: BIOS/firmware updates via vendor tools (Dell idrac, HP ssa). (6) Application-specific configuration from Git-backed Ansible roles. (7) Verification: InSpec/Serverspec compliance checks. Total provisioning time: <30 minutes for bare metal.

15. How do you use Python for Linux administration tasks? Give examples.

In Simple Terms:

Python is a programming language that helps us automate repetitive tasks — instead of manually checking 500 servers one by one, a Python script can do it in seconds.

Technical Context:

Python with libraries like paramiko (SSH), fabric, subprocess, psutil, and custom APIs allows sophisticated automation and monitoring solutions.

Sample Answer:

Examples I've built: (1) Server health aggregator using psutil and paramiko — collects CPU/mem/disk from fleet, feeds Prometheus. (2) Automated incident correlation — parses logs via regex, correlates with monitoring alerts using Python-Jira API. (3) PTP time sync verifier — reads /dev/ptp timestamps, calculates drift, alerts on threshold breach. (4) DPDK hugepage configurator — validates NUMA topology and configures hugepages per socket. (5) Change management validator — pre/post change comparison of sysctl, network routes, process list. All stored in Git with unit tests.

SECTION 2: 30 REAL-WORLD PRODUCTION SCENARIOS

Each scenario includes: Layman explanation, Technical explanation, Diagnosis steps with commands, Resolution steps, and command output interpretation.

Scenario 1: Server Load Average is Extremely High

In Simple Terms:

Think of load average like a traffic jam on a highway. If cars (processes) are waiting to get through, the highway (CPU) is overloaded. We need to find which car is causing the jam.

Technical Background:

Load average represents the number of processes in runnable or uninterruptible sleep state. A 15-minute load average > number of CPUs indicates sustained overload.

Diagnosis Commands:

```
uptime  # Check load average: 1min, 5min, 15min
w       # See who is logged in and their load
top     # Real-time process view sorted by CPU
vmstat 1 5   # Check r (run queue), b (blocked), wa (iowait)
mpstat -P ALL 1   # Per-CPU utilization
ps aux --sort=-%cpu | head -20   # Top CPU consumers
pidstat -u 1 5   # Per-process CPU stats over time
```

Reading the Output:

vmstat output: r=15 means 15 processes waiting for CPU. wa=60 means 60% time waiting for I/O — indicates disk bottleneck. b=5 means 5 blocked processes.

Resolution Steps:

```
# If CPU bound: identify and renice or kill offending process
renice +10 -p <PID>   # Lower priority
kill -15 <PID>         # Graceful kill
# If I/O wait is high: check disk
iostat -x 1   # Look for %util near 100 on a device
iotop -a      # Find top I/O consuming processes
# If swap is causing it:
free -h       # Check swap usage
swapon -s     # List swap devices
vmstat -s | grep swap
```

Scenario 2: Server Runs Out of Memory — OOM Killer Fires

In Simple Terms:

When your computer runs out of memory (RAM), it's like a kitchen running out of counter space. The OS has to make tough decisions: it picks the application using the most space and forcibly shuts it down to free up room.

Technical Background:

The Linux OOM (Out of Memory) Killer is invoked when the kernel cannot satisfy memory allocation requests. It selects a process based on oom_score and kills it.

Diagnosis Commands:

```
dmesg -T | grep -i 'oom\|killed process'   # Find OOM events
cat /var/log/messages | grep -i oom          # System log OOM messages
free -h            # Current memory state
cat /proc/meminfo # Detailed memory breakdown
smem -r -k | head -20  # Sorted memory consumers
ps aux --sort=-%mem | head -20
```

Reading the Output:

dmesg output: 'Out of memory: Kill process 12345 (java) score 892 or sacrifice child' — shows which process was killed and its OOM score (higher = more likely to be killed).

Resolution Steps:

```
# Immediate: add swap as emergency buffer
dd if=/dev/zero of=/swapfile bs=1G count=8
chmod 600 /swapfile && mkswap /swapfile && swapon /swapfile
# Tune OOM score for critical processes (protect them)
echo -500 > /proc/<critical_pid>/oom_score_adj  # -1000 = never kill
# Tune OOM score for expendable processes (sacrifice first)
echo 1000 > /proc/<pid>/oom_score_adj
# Long term: add RAM or fix memory leak
valgrind --leak-check=full ./app   # Detect memory leaks
```

Scenario 3: Network Interface is Dropping Packets

In Simple Terms:

Imagine sending letters by mail, but the post office is losing some of them. In networking, packet drops mean data is being lost, which causes retransmissions, slowing everything down — critical in trading.

Technical Background:

Packet drops can occur at NIC ring buffer level (hardware), kernel receive queue (softirq backlog), or application socket buffer. Each has different causes and solutions.

Diagnosis Commands:

```
ip -s link show eth0   # TX/RX stats including errors/drops
ethtool -S eth0 | grep -i drop   # NIC-level drop counters
netstat -su   # Socket-level statistics
```

```
cat /proc/net/softnet_stat   # Softirq stats (col3 = drops)
sar -n DEV 1 5   # Network device stats over time
watch -n1 'ethtool -S eth0 | grep -E "missed|drop|error"'
```

Reading the Output:

ethtool -S: 'rx_missed_errors: 45231' — NIC ring buffer overflow. /proc/net/softnet_stat col3 non-zero = kernel backlog drops. 'ip -s': RX errors count increasing = hardware or driver issue.

Resolution Steps:

```
# Increase NIC ring buffer size
ethtool -g eth0    # Check current/max ring buffer size
ethtool -G eth0 rx 4096 tx 4096   # Increase ring buffers
# Increase kernel receive backlog
sysctl -w net.core.netdev_max_backlog=10000
sysctl -w net.core.rmem_max=268435456
# Pin NIC interrupts to specific CPUs
cat /proc/interrupts | grep eth0    # Find IRQ number
echo 4 > /proc/irq/<IRQ>/smp_affinity   # Assign to CPU 2
# Enable RSS (Receive Side Scaling)
ethtool -L eth0 combined 8   # Use 8 queues
```

Scenario 4: Disk Space is Full — System Alerts

In Simple Terms:

A full disk is like a full filing cabinet — you can't store any more documents, and the system may crash or applications may fail. We need to find what's taking up space and clear it.

Technical Background:

A full filesystem causes application write failures, log rotation failures, and potential data corruption. Often caused by log accumulation, core dumps, or large temporary files.

Diagnosis Commands:

```
df -h   # Disk space usage per filesystem
df -i   # Inode usage (filesystem can be 'full' with inodes even if space available)
du -sh /* 2>/dev/null | sort -rh | head -20   # Top space consumers
du -sh /var/log/* | sort -rh | head -10    # Check logs
find / -name '*.log' -size +1G 2>/dev/null   # Large log files
find / -name 'core' -size +100M 2>/dev/null   # Core dumps
lsof | grep deleted | awk '{print $7, $1, $2}' | sort -rn | head   # Deleted but open
files
```

Reading the Output:

lsof deleted files: Shows files that were deleted but are still held open by processes — disk space not freed until process closes or is killed.

Resolution Steps:

```
# Clear old logs safely
journalctl --vacuum-size=500M   # Trim systemd journal
find /var/log -name '*.gz' -mtime +30 -delete   # Old compressed logs
> /var/log/large_app.log   # Truncate (don't delete open log files)
# Clean package cache
yum clean all   # RHEL
# Remove core dumps
find / -name 'core.*' -type f -delete
# For deleted-but-open files: restart the process holding it
lsof | grep deleted | awk '{print $2}' | sort -u | xargs kill -HUP
# Add disk or expand LVM
lvextend -L +50G /dev/vg0/lv_data && xfs_growfs /data
```

Scenario 5: SSH Connection Refused or Hanging

In Simple Terms:

SSH is how we remotely access servers. If SSH is broken, it's like being locked out of your own house. We need different ways to get in and fix the door.

Technical Background:

SSH failures can be caused by sshd service failure, firewall rules, host-based access controls, authentication issues, or TCP connection problems.

Diagnosis Commands:

```
telnet <server> 22   # Test TCP port 22 connectivity
nc -zv <server> 22   # Alternative port test
ssh -vvv user@server 2>&1 | head -50   # Verbose SSH debug
# From the server (via console/IPMI):
systemctl status sshd   # Is sshd running?
journalctl -u sshd -n 50   # Recent sshd logs
ss -tlnp | grep :22   # Is sshd listening?
iptables -L -n | grep 22   # Firewall rules
cat /etc/hosts.deny   # TCP wrappers
```

Reading the Output:

ssh -vvv: Shows exactly where connection hangs — 'debug1: connect to address port 22: Connection refused' = sshd not running. Hanging after 'debug1: SSH2_MSG_SERVICE_REQUEST sent' = authentication issue.

Resolution Steps:

```
# Via IPMI console if SSH is broken:
systemctl restart sshd
# Fix firewall if blocking:
firewall-cmd --add-port=22/tcp --permanent && firewall-cmd --reload
# Fix SELinux if blocking:
ausearch -m avc -ts recent | grep sshd   # Check SELinux denials
```

```
setsebool -P nis_enabled 1    # Common fix
# Fix too many auth failures (fail2ban):
fail2ban-client set sshd unbanip <your_ip>
# Check MaxSessions/MaxStartups in /etc/ssh/sshd_config
grep -E 'MaxSessions|MaxStartups' /etc/ssh/sshd_config
```

Scenario 6: Process is in Uninterruptible Sleep (D State)

In Simple Terms:

A process in 'D state' is stuck waiting for something — usually a disk or NFS operation — and cannot be interrupted, even with a kill command. It's like an employee frozen mid-task who can't hear you calling them.

Technical Background:

Uninterruptible sleep (D state) means a process is waiting in kernel code that cannot be interrupted. Most common cause: I/O wait on hung NFS mount, bad disk, or storage timeout.

Diagnosis Commands:

```
ps aux | grep ' D '    # Find processes in D state
cat /proc/<PID>/status | grep State    # Confirm D state
cat /proc/<PID>/wchan    # What kernel function it's waiting on
cat /proc/<PID>/stack    # Kernel stack trace
ls -la /proc/<PID>/fd    # What files it has open
strace -p <PID>    # Trace syscalls (may not work in D state)
dmesg | tail -50    # Check for NFS/disk errors
```

Reading the Output:

wchan output: 'nfs_do_read' = stuck on NFS. 'jbd2_log_wait_commit' = stuck on journal flush. Stack trace shows exact kernel path.

Resolution Steps:

```
# Cannot kill D state process — must fix the underlying I/O
# If NFS mount hung:
umount -f -l /mnt/nfs    # Force lazy unmount
# If disk I/O hung:
dmesg | grep 'I/O error'
smartctl -a /dev/sdb    # Check disk health
# Fix NFS server side, then remount
mount <nfs_server>:/export /mnt/nfs
# If cannot fix — reboot is often only option for stuck D processes
# Prevent: use 'soft' NFS mounts with timeo for production
# /etc/fstab: nfs_server:/export /mnt nfs soft,timeo=30,retrans=2 0 0
```

Scenario 7: Trading Application Missing Heartbeat — High Latency Alert

In Simple Terms:

Trading applications send heartbeat messages to say 'I'm alive.' If these stop, something is wrong. We need to quickly find if it's a network problem, the application itself, or the server.

Technical Background:

Heartbeat failures in trading systems can indicate application thread hang, GC pause, network issue, or OS-level problem (high CPU, memory pressure, scheduling jitter).

Diagnosis Commands:

```
ping -c 5 <target>   # Basic connectivity
traceroute <target>   # Path and latency per hop
ss -tnp | grep <port>   # Check established connections
netstat -s | grep 'segments retransmited'   # TCP retransmits
# Application level:
jstack <PID>   # Java thread dump (find blocked/waiting threads)
strace -p <PID> -e trace=network   # System call trace
# OS level:
cat /proc/<PID>/schedstat   # CPU scheduling stats
perf stat -p <PID> sleep 10   # Performance counters
```

Reading the Output:

jstack output: Threads in 'BLOCKED' state on a lock = application deadlock. 'waiting for GC' in thread names = GC pause causing heartbeat miss.

Resolution Steps:

```
# GC pause fix: tune JVM GC
# Add to JVM flags: -XX:+UseG1GC -XX:MaxGCPauseMillis=10
# Thread deadlock: restart application, fix code
# Network issue: check ethtool, increase socket buffers
sysctl -w net.core.rmem_default=16777216
sysctl -w net.core.wmem_default=16777216
# OS scheduling jitter: enable CPU isolation
# Add to /etc/default/grub GRUB_CMDLINE_LINUX:
# isolcpus=2,3,4,5 nohz_full=2,3,4,5 rcu_nocbs=2,3,4,5
grub2-mkconfig -o /boot/grub2/grub.cfg && reboot
```

Scenario 8: File Descriptor Limit Reached

In Simple Terms:

Every open file, socket, or connection uses a 'file descriptor.' When you hit the limit, the application can't open any more connections or files — like a switchboard operator who can only handle a fixed number of calls simultaneously.

Technical Background:

Linux has per-process and system-wide file descriptor limits. Trading systems with many network connections can exhaust limits. 'Too many open files' error is the symptom.

Diagnosis Commands:

```
ulimit -n   # Current shell's file descriptor limit
cat /proc/sys/fs/file-max   # System-wide max
lsof -p <PID> | wc -l   # FDs used by specific process
cat /proc/<PID>/limits | grep 'open files'   # Actual limits for PID
lsof | wc -l   # Total open files system-wide
cat /proc/sys/fs/file-nr   # Used/free/max file descriptors
```

Reading the Output:

file-nr output: '45000 0 100000' = 45000 used out of 100000 max. /proc/<PID>/limits: 'Max open files: 1024' — very low for production.

Resolution Steps:

```
# Immediate: increase for current session
ulimit -n 65536
# Per-service: /etc/systemd/system/<service>.d/override.conf
[Service]
LimitNOFILE=1000000
systemctl daemon-reload && systemctl restart <service>
# System-wide persistent: /etc/security/limits.conf
* soft nofile 100000
* hard nofile 200000
# Also in /etc/sysctl.conf:
fs.file-max = 1000000
sysctl -p
```

Scenario 9: Zombie Processes Accumulating

In Simple Terms:

Zombie processes are like employees who have finished and quit their jobs but whose termination paperwork was never processed. They don't consume CPU but do waste process table entries. Too many zombies can prevent new processes from starting.

Technical Background:

A zombie (Z state) process has completed but its parent hasn't called wait() to collect its exit status. The fix is always in the parent process, not the zombie.

Diagnosis Commands:

```
ps aux | grep ' Z '   # Find zombies
ps -eo pid,ppid,stat,cmd | grep ' Z '   # With parent PID
pstree -p | grep defunct   # Zombie tree
# Find parent of zombie:
```

```
ps -o ppid= -p <ZOMBIE_PID>
# Check parent process health
strace -p <PARENT_PID> -e wait4   # Is parent calling wait()?
```

Reading the Output:

ps output: 'Z' in STAT column = zombie. PPID shows the parent. If parent is PID 1 (init/systemd), parent already died and systemd should reap it.

Resolution Steps:

```
# Send SIGCHLD to parent to trigger wait()
kill -CHLD <PARENT_PID>
# If parent itself is stuck: kill parent (zombies auto-reparent to init and get reaped)
kill -9 <PARENT_PID>
# If parent is a service: restart it
systemctl restart <service>
# Long-term fix: code must properly handle SIGCHLD and call waitpid()
# Monitor zombie count:
watch -n5 'ps aux | grep -c " Z "'
```

Scenario 10: NFS Mount Becomes Stale or Unresponsive

In Simple Terms:

NFS (Network File System) lets servers share files over a network. A stale NFS mount is like a map pointing to a building that no longer exists — your computer tries to access files but the path leads nowhere, causing hangs.

Technical Background:

NFS mounts can become stale if the NFS server reboots, network interruption occurs, or the export is removed. Hard mounts hang indefinitely; soft mounts fail with an error.

Diagnosis Commands:

```
mount | grep nfs   # List NFS mounts
df -h 2>&1 | grep -v Filesystem   # Hanging here = stale NFS
nfsstat -m   # NFS mount statistics
showmount -e <nfs_server>   # Check server exports
rpcinfo -p <nfs_server>   # Check RPC services on server
cat /proc/mounts | grep nfs
# Check for processes stuck on NFS:
ps aux | grep ' D '
```

Reading the Output:

df hanging = stale mount. showmount failing = NFS server unreachable. rpcinfo failure = portmapper/rpc.mountd not running on server.

Resolution Steps:

```
# Lazy unmount (detaches mount point, lets existing FDs drain)
umount -f -l /mnt/nfs
# If processes stuck in D state on NFS:
fuser -km /mnt/nfs   # Kill all processes using mount
umount -f /mnt/nfs
# Fix NFS server then remount
systemctl restart nfs-server   # On NFS server
mount <server>:/export /mnt/nfs
# Prevention: use soft mounts with timeout
# In /etc/fstab: server:/export /mnt nfs soft,intr,timeo=30,retrans=2,_netdev 0 0
# Or use autofs for automounting
```

Scenario 11: Kernel Panic Recovery

In Simple Terms:

A kernel panic is like the computer's brain having a seizure — the OS hits a critical error and completely freezes or reboots. Like a car's engine seizing up, it needs diagnosis after the fact using 'black box' data.

Technical Background:

Kernel panics produce a crash dump (via kdump/kexec) containing a vmcore file. Analysis with crash utility reveals the exact kernel thread, call stack, and memory state at time of panic.

Diagnosis Commands:

```
# After reboot:
ls /var/crash/   # kdump saves core here
crash /usr/lib/debug/lib/modules/$(uname -r)/vmlinux /var/crash/<date>/vmcore
# Inside crash utility:
bt   # Backtrace of crashing thread
log  # Kernel message log
ps   # Process list at time of crash
vm   # Memory usage
# Check /var/log/messages before crash time
dmesg -T | grep -i 'error\|panic\|call trace' | tail -50
```

Reading the Output:

crash bt output: Shows exact kernel function stack that caused the panic. Common patterns: null pointer dereference in driver, memory corruption, divide-by-zero in kernel module.

Resolution Steps:

```
# Ensure kdump is configured (before next panic):
yum install kexec-tools crash
systemctl enable kdump && systemctl start kdump
# /etc/kdump.conf: path /var/crash
# Reserve memory: add crashkernel=256M to GRUB kernel params
# After analysis: update/remove faulty kernel module
modprobe -r <faulty_module>
```

```
# Blacklist module: echo 'blacklist <module>' > /etc/modprobe.d/blacklist.conf
# Update kernel if bug known fixed:
yum update kernel
```

Scenario 12: SELinux Blocking Application Access

In Simple Terms:

SELinux is a security guard that checks every file access request. Sometimes it's overly strict and blocks legitimate applications. The fix is to teach the security guard the new rules — not disable it entirely.

Technical Background:

SELinux uses Mandatory Access Control (MAC) with type enforcement. When an application tries to access a resource it doesn't have a policy rule for, SELinux denies it and logs an AVC denial.

Diagnosis Commands:

```
sestatus   # SELinux status and mode
ausearch -m avc -ts recent   # Recent AVC denials
ausearch -m avc -ts recent | audit2why   # Explain why denied
journalctl | grep 'SELinux is preventing'
ls -Z /path/to/file   # File's SELinux context
ps -eZ | grep <process>   # Process SELinux context
sealert -a /var/log/audit/audit.log   # Human-readable summary
```

Reading the Output:

ausearch: 'avc: denied { read } for pid=1234 comm="app" name="config.conf" scontext=system_u:system_r:httpd_t tcontext=staff_u:object_r:user_home_t' — process type doesn't have permission on file type.

Resolution Steps:

```
# Option 1: Fix file context (preferred)
chcon -t httpd_sys_content_t /path/to/file   # Temporary
semanage fcontext -a -t httpd_sys_content_t '/path/to/dir(/.*)?'
restorecon -Rv /path/to/dir   # Apply permanently
# Option 2: Generate custom policy module
ausearch -m avc -ts recent | audit2allow -M myapp
semodule -i myapp.pp
# Option 3: Set booleans for common scenarios
getsebool -a | grep httpd   # List relevant booleans
setsebool -P httpd_can_network_connect 1
# NEVER: setenforce 0 in production (disables SELinux)
```

Scenario 13: Clock Drift — NTP/PTP Synchronization Issue

In Simple Terms:

In trading, all servers need to have the exact same time — microsecond differences matter for trade timestamps. Clock drift is like your watch slowly running fast or slow. We need to identify and correct it immediately.

Technical Background:

In financial environments, clock accuracy is required for regulatory reporting (MiFID II requires microsecond timestamps). PTP (IEEE 1588) with hardware timestamping achieves sub-microsecond accuracy.

Diagnosis Commands:

```
timedatectl status   # System time and sync status
chronyc tracking   # Chrony sync status and offset
chronyc sources -v   # Time sources and their quality
chronyc sourcestats   # Statistics for each source
# For PTP:
ptp4l -m -i eth0   # Check PTP offset
phc2sys -s eth0 -w -m   # Sync system clock to PTP
# Check hardware capability:
ethtool -T eth0   # Hardware timestamping support
```

Reading the Output:

chronyc tracking: 'System time: 0.000012345 seconds fast' = 12 microseconds fast. 'Reference ID: PTP' = synced to PTP. Offset > 1ms = problem.

Resolution Steps:

```
# Immediate: step clock (if offset > 1 second)
chronyc makestep
# Force re-sync:
systemctl restart chronyd
chronyc burst 4/4
# For PTP setup on Solarflare/Mellanox NIC:
ptp4l -f /etc/ptp4l.conf -i eth0 -m &
phc2sys -s eth0 -c CLOCK_REALTIME -w -m &
# Configure chrony to use PTP hardware clock:
# In /etc/chrony.conf: refclock PHC /dev/ptp0 poll 0 dpoll -2 offset 0
```

Scenario 14: Slow DNS Resolution Impacting Application Performance

In Simple Terms:

DNS is like a phone book. If looking up a phone number takes 5 seconds, every call you make starts with a 5-second delay. In trading, slow DNS means slower order routing and connection establishment.

Technical Background:

DNS resolution latency can be caused by resolver misconfiguration, upstream DNS server issues, missing local cache, or ndots configuration causing excessive search domain lookups.

Diagnosis Commands:

```
time dig google.com @<dns_server>   # Measure DNS query time
time nslookup tradingsystem.ms.com   # Alternative test
cat /etc/resolv.conf   # DNS server config
cat /etc/nsswitch.conf   # Name resolution order
tcpdump -i eth0 port 53   # Capture DNS traffic
systemd-resolve --statistics   # Cache stats (RHEL 8+)
dig +trace hostname.   # Full resolution trace
```

Reading the Output:

dig with @resolver: If response time > 50ms = DNS server slow. 'SERVFAIL' = server error. Multiple queries in tcpdump for one hostname = ndots search domain loop.

Resolution Steps:

```
# Add local caching resolver
yum install dnsmasq
systemctl enable --now dnsmasq
# In /etc/resolv.conf: nameserver 127.0.0.1
# Reduce ndots to prevent excessive searches
# In /etc/resolv.conf: options ndots:1 timeout:1 attempts:2
# Pre-cache critical hostnames
nscd   # Or use unbound as local resolver with prefetch
# If DNS server is overloaded: check server load
rndc stats   # BIND stats
tail /var/named/data/named_stats.txt
```

Scenario 15: Filesystem Corruption After Unclean Shutdown

In Simple Terms:

When a server loses power suddenly, it's like knocking over a stack of papers mid-sort. Some files may be partially written and corrupted. We need to check and repair the 'filing system.'

Technical Background:

XFS and ext4 are journaling filesystems that recover most corruption on mount. But severe corruption may require fsck. Never run fsck on a mounted filesystem.

Diagnosis Commands:

```
dmesg | grep -i 'xfs\|ext4\|error\|corrupt'
journalctl -b-1 | grep -i error   # Logs from previous boot
xfs_info /dev/sda1   # XFS filesystem info
xfs_db -r /dev/sda1 -c 'check'   # XFS check (read-only)
tune2fs -l /dev/sda1   # ext4 info including error count
e2fsck -n /dev/sda1   # Dry-run ext4 check (don't fix)
```

Reading the Output:

dmesg: 'XFS: Internal error XFS_WANT_CORRUPTED_RETURN' or 'EXT4-fs error (device sda1): ext4_find_entry: reading directory' = corruption detected.

Resolution Steps:

```
# Boot to rescue mode or unmount filesystem
# For XFS:
xfs_repair /dev/sda1   # Repair XFS
xfs_repair -L /dev/sda1   # Force log zeroing if journal corrupt
# For ext4:
e2fsck -fp /dev/sda1   # Automatic fix, no prompts
e2fsck -fy /dev/sda1   # Fix all, yes to all prompts
# If LVM: take snapshot before repair
lvcreate -L 10G -s -n snap /dev/vg0/lv_data
# Prevention: use UPS, configure 'nobarrier' only on reliable storage
```

Scenario 16: iptables/firewalld Rules Blocking Traffic

In Simple Terms:

Firewall rules are like bouncers at a club — they decide who gets in and who doesn't. Sometimes a bouncer is too strict and blocks legitimate guests. We need to find and fix the specific rule that's causing problems.

Technical Background:

iptables processes packets through chains (INPUT, OUTPUT, FORWARD) with rules matched top-to-bottom. First matching rule wins. firewalld is a high-level wrapper around iptables/nftables.

Diagnosis Commands:

```
iptables -L -n -v --line-numbers   # All rules with counters
iptables -L INPUT -n -v   # INPUT chain
firewall-cmd --list-all   # firewalld current rules
# Test specific connection:
nc -zv <host> <port>   # Check if port reachable
tcpdump -i eth0 host <target> and port <port>   # Capture traffic
# Trace packet through rules:
iptables -t raw -A PREROUTING -s <src_ip> -j TRACE
xtables-monitor --trace   # Monitor traced packets
```

Reading the Output:

iptables -L with -v: Packets/bytes counter increases on a rule = that rule is matching. A REJECT/DROP rule with high counter = likely culprit.

Resolution Steps:

```
# Allow specific port temporarily (for testing):
```

```
iptables -I INPUT 1 -p tcp --dport 8080 -j ACCEPT
# Permanent via firewalld:
firewall-cmd --add-port=8080/tcp --permanent
firewall-cmd --reload
# Allow specific service:
firewall-cmd --add-service=https --permanent
# Remove blocking rule by line number:
iptables -D INPUT 5   # Delete rule 5
# Save iptables rules:
iptables-save > /etc/sysconfig/iptables
# For nftables (RHEL 8+):
nft list ruleset
```

Scenario 17: SSL/TLS Certificate Expiry Causing Service Failure

In Simple Terms:

SSL certificates are like ID cards for websites — they prove identity and enable encryption. When they expire, browsers and systems refuse to connect, like being denied entry because your ID is expired.

Technical Background:

Expired certificates cause TLS handshake failures. In trading, this can break connectivity to exchanges, brokers, or inter-service communication.

Diagnosis Commands:

```
# Check certificate expiry:
openssl s_client -connect host:443 </dev/null 2>/dev/null | openssl x509 -noout -dates
# Check local certificate files:
openssl x509 -in /etc/ssl/certs/app.crt -noout -text | grep 'Not After'
# Check all certs expiring in 30 days:
find /etc/ssl -name '*.crt' -exec openssl x509 -noout -in {} -checkend 2592000 \;
# Via curl:
curl -vI https://host 2>&1 | grep -E 'expire|SSL|error'
```

Reading the Output:

openssl: 'notAfter=Jan 15 00:00:00 2024 GMT' = expired. curl: 'SSL certificate problem: certificate has expired' = TLS handshake failure.

Resolution Steps:

```
# Generate new CSR:
openssl req -new -key server.key -out server.csr
# If Let's Encrypt:
certbot renew --cert-name hostname
certbot renew --force-renewal
# If internal CA: submit CSR to CA team
# After new cert received:
openssl verify -CAfile ca.crt new.crt   # Verify chain
cp new.crt /etc/ssl/certs/app.crt
```

```
systemctl restart nginx   # Reload service
# Set up monitoring:
# Prometheus blackbox_exporter with ssl_expiry probe
# Or: script in cron to email when cert < 30 days
```

Scenario 18: Runaway Process Consuming All CPU

In Simple Terms:

A runaway process is like an employee who keeps working at full speed without stopping, hogging all shared resources and making everything else slow. We need to identify and deal with them.

Technical Background:

A process consuming excessive CPU can be caused by an infinite loop, inefficient algorithm under load, or a process in a tight spin-wait loop (common in kernel bypass applications).

Diagnosis Commands:

```
top -b -n1 | head -30   # Snapshot top processes
htop   # Interactive, color-coded view
pidstat -u 1 5   # Per-process CPU over time
ps -eo pid,ppid,%cpu,%mem,cmd --sort=-%cpu | head -10
# Profile the process:
perf top -p <PID>   # Real-time profiling
perf record -g -p <PID> sleep 30 && perf report
# Flame graph:
perf script | stackcollapse-perf.pl | flamegraph.pl > flame.svg
```

Reading the Output:

perf top: Shows which functions are consuming CPU. If spin_do_poll or busy_wait_for_event = intentional busy polling (normal for DPDK). If it's in application code = bug.

Resolution Steps:

```
# Immediate mitigation: reduce CPU priority
renice +15 -p <PID>   # Background priority
# Limit CPU via cgroups:
cgcreate -g cpu:/limited
echo 50000 > /sys/fs/cgroup/cpu/limited/cpu.cfs_quota_us   # 50% of one CPU
cgclassify -g cpu:/limited <PID>
# Or use taskset to pin to specific core away from critical processes:
taskset -cp 7 <PID>   # Move to CPU 7
# If bug: kill and fix application
kill -9 <PID>
# Investigate with perf report to find the hot function
```

Scenario 19: Ansible Playbook Fails on Multiple Hosts

In Simple Terms:

Ansible is our automation tool for managing many servers. When a playbook fails, it's like a recipe that works at home but fails when cooking for 100 people. We need to find which step failed and why.

Technical Background:

Ansible failures can be idempotency issues, version mismatches, connectivity problems, vault encryption issues, or task timeouts. Understanding the error and affected hosts is key.

Diagnosis Commands:

```
# Run with verbose output:
ansible-playbook -i inventory site.yml -vvv 2>&1 | tee /tmp/ansible_debug.log
# Run on single failed host:
ansible-playbook -i inventory site.yml --limit failed_host -vvv
# Check syntax first:
ansible-playbook site.yml --syntax-check
# Dry run:
ansible-playbook site.yml --check --diff
# Test connectivity:
ansible all -m ping -i inventory
```

Reading the Output:

Ansible output: 'FAILED!' in red with error message. 'fatal: [host]: UNREACHABLE!' = SSH/connectivity issue. 'fatal: [host]: FAILED! => {"msg": "Authentication failure"}' = SSH key issue.

Resolution Steps:

```
# Fix SSH key issues:
ssh-copy-id -i ~/.ssh/ansible.pub user@host
# Fix become (sudo) issues:
ansible host -m shell -a 'sudo whoami' --become
# Check /etc/sudoers on target for NOPASSWD
# Fix vault decryption:
ansible-vault decrypt secrets.yml
# Re-encrypt with correct password:
ansible-vault rekey secrets.yml
# Fix variable precedence issues:
ansible-playbook site.yml -e 'debug_var=true' --check
# Run failed tasks only:
ansible-playbook site.yml --start-at-task='failing task name'
```

Scenario 20: Multicast Traffic Not Being Received by Trading Application

In Simple Terms:

Market data in trading is often delivered via multicast — like a radio broadcast. If the radio (server) isn't tuned to the right frequency (multicast group) or the signal path is blocked, you don't receive any data.

Technical Background:

Multicast requires proper IGMP group membership, correct routing, NIC configuration, and no firewall blocking. Issues can be at NIC, OS, or network switch level.

Diagnosis Commands:

```
ip maddr show   # Current multicast group memberships
netstat -g   # Multicast group memberships
tcpdump -i eth0 host 239.1.1.1   # Capture multicast traffic
ss -u -a | grep <port>   # UDP socket state
ethtool -k eth0 | grep rx-all   # Promiscuous mode
# Check IGMP:
tcpdump -i eth0 igmp   # IGMP join/leave messages
cat /proc/net/igmp   # IGMP memberships
```

Reading the Output:

tcpdump showing no traffic on multicast group = traffic not reaching server. tcpdump on switch port showing traffic = network is delivering but OS not receiving. ip maddr not showing group = application hasn't joined.

Resolution Steps:

```
# Verify NIC can receive multicast:
ethtool -l eth0   # Check combined queues
# Allow multicast in firewall:
iptables -A INPUT -d 239.0.0.0/8 -j ACCEPT
# Add static multicast route if needed:
ip route add 239.0.0.0/8 dev eth0
# Enable multicast on interface:
ip link set eth0 multicast on
# Test reception with socat:
socat UDP4-RECVFROM:5000,ip-add-membership=239.1.1.1:eth0 -
# If NIC filtering: check/disable igmp snooping on switch VLAN
# Application: ensure SO_REUSEPORT and proper bind to INADDR_ANY
```

Scenario 21: Server Cannot Boot After Kernel Update

In Simple Terms:

After installing a new kernel (the OS core), the server fails to start. It's like replacing a car engine with the wrong model — the car won't start. We need to boot with the old engine and investigate.

Technical Background:

Kernel update boot failures can be caused by initramfs missing modules, incompatible drivers, GRUB misconfiguration, or hardware not supported by new kernel.

Diagnosis Commands:

```
# From GRUB menu: select previous kernel
# Boot to rescue mode: append 'rescue' to kernel params in GRUB
# After booting old kernel:
uname -r   # Current kernel version
rpm -q kernel   # Installed kernels
cat /boot/grub2/grub.cfg | grep menuentry   # GRUB menu entries
# Check new kernel initramfs:
lsinitrd /boot/initramfs-<new_kernel>.img | head -50
dmesg | head -100   # Boot messages from failed kernel
```

Reading the Output:

dmesg: 'Failed to mount root filesystem' = initramfs missing driver. 'kernel panic - not syncing' = critical driver missing. GRUB: New kernel entry present but pointing to wrong initramfs.

Resolution Steps:

```
# Boot with old kernel via GRUB (temporary)
# Regenerate initramfs for new kernel:
dracut -f /boot/initramfs-<new_kernel>.img <new_kernel>
# Rebuild GRUB:
grub2-mkconfig -o /boot/grub2/grub.cfg
# Set default kernel (revert to old):
grub2-set-default 'Red Hat Enterprise Linux (old_kernel) 8.x'
# Or by index:
grub2-set-default 1
# Install missing drivers for new kernel:
yum install kernel-modules-extra-<new_kernel>
# Prevent automatic kernel updates in critical systems:
yum install python3-dnf-plugin-versionlock
yum versionlock kernel-$(uname -r)
```

Scenario 22: High TCP Retransmission Rate

In Simple Terms:

TCP retransmissions are like resending a letter because the postal service lost the original. In trading, retransmissions add latency and can cause connection drops. We need to find if it's network hardware, configuration, or congestion.

Technical Background:

TCP retransmissions indicate packet loss. In LAN environments, packet loss should be near zero. Retransmits can be caused by CRC errors (NIC/cable), congestion (switch buffer overflow), or misconfigured network settings.

Diagnosis Commands:

```
netstat -s | grep -i retransmit   # System-wide retransmit count
ss -ti dst <host>   # TCP info including retransmits for connection
ethtool -S eth0 | grep -i error   # NIC hardware errors
ip -s link show eth0   # RX errors/drops
```

```
tcpdump -i eth0 -nn 'tcp[13] & 4 != 0'    # Capture RST packets
# Monitor over time:
sar -n TCP 1 10    # TCP stats per second
nstat -az TcpRetransSegs    # Retransmit counter
```

Reading the Output:

netstat -s: 'X segments retransmitted' increasing rapidly = active problem. ethtool: 'rx_crc_errors: 1234' = cable/NIC hardware problem. ss -ti: 'retrans:5 lost:5' for a connection = that path has packet loss.

Resolution Steps:

```
# If hardware errors: replace cable or NIC
ethtool eth0 | grep Speed    # Check negotiated speed
ethtool -s eth0 speed 10000 duplex full autoneg off    # Force speed
# If switch congestion: enable flow control
ethtool -A eth0 rx on tx on    # Enable pause frames
# Tune TCP retransmission timeout:
sysctl -w net.ipv4.tcp_retries2=5    # Fail faster
# If buffer bloat: tune socket buffers
sysctl -w net.ipv4.tcp_rmem='4096 87380 16777216'
sysctl -w net.ipv4.tcp_wmem='4096 65536 16777216'
# Monitor with continuous watch:
watch -n1 'netstat -s | grep retransmit'
```

Scenario 23: /tmp Filesystem Full — Applications Failing

In Simple Terms:

/tmp is a scratch pad for temporary files. When it's full, applications that need to write temporary data fail — like a whiteboard being completely covered with no space to write new notes.

Technical Background:

/tmp can be on the root filesystem, a dedicated partition, or tmpfs (RAM-backed). Full /tmp can cause compilation failures, application crashes, and login failures.

Diagnosis Commands:

```
df -h /tmp    # Check /tmp usage
mount | grep tmp    # What type is /tmp mounted as
du -sh /tmp/* 2>/dev/null | sort -rh | head -10    # Biggest items in /tmp
lsof /tmp    # Files in /tmp held open
find /tmp -type f -atime +7    # Files not accessed in 7 days
ls -la /tmp    # List tmp directory contents
```

Reading the Output:

df: '/tmp 100% full'. du showing large files owned by root or specific user. lsof showing open tmp files from running processes.

Resolution Steps:

```
# Clear old files immediately:
find /tmp -type f -atime +1 -delete   # Files older than 1 day
find /tmp -name '*.tmp' -delete
# If specific application owns large files: restart it
# Kill orphan processes using /tmp:
fuser -k /tmp   # Kill processes using /tmp (careful!)
# If tmpfs: increase size
mount -o remount,size=4G /tmp
# Persistent: in /etc/fstab: tmpfs /tmp tmpfs defaults,size=4G 0 0
# Set up automatic cleanup:
systemctl enable --now systemd-tmpfiles-clean.timer
# In /etc/tmpfiles.d/tmp.conf:
# D /tmp 1777 root root 7d
```

Scenario 24: Systemd Service Keeps Restarting (Crash Loop)

In Simple Terms:

A crash loop is like an employee who shows up, immediately faints, gets taken away, comes back, and faints again — repeatedly. We need to find out why they keep fainting.

Technical Background:

Systemd's restart policy causes services to restart on failure. If a service crashes immediately after start, it enters a rapid restart loop. Understanding the crash requires logs and crash dumps.

Diagnosis Commands:

```
systemctl status <service>   # Current state and recent log
journalctl -u <service> -n 100 --no-pager   # Last 100 log lines
journalctl -u <service> -f   # Follow logs in real time
systemctl show <service> | grep -E 'Restart|NRestarts|ExecStart'
# Check for core dumps:
coredumpctl list   # List core dumps
coredumpctl info <PID>   # Core dump details
coredumpctl debug <PID>   # Debug with gdb
```

Reading the Output:

journalctl: Shows the actual error before crash. 'Segmentation fault' or 'Aborted (core dumped)' = application crash. 'Address already in use' = port conflict. systemctl show: NRestarts=47 = 47 restart attempts.

Resolution Steps:

```
# Identify crash cause from logs/coredump first
# Stop crash loop temporarily:
systemctl stop <service>
# Disable auto-restart temporarily while debugging:
systemctl edit <service>   # Add: [Service] Restart=no
# Fix common causes:
```

```
# Port conflict: find and kill process on same port
ss -tlnp | grep <port>
fuser -k <port>/tcp
# Missing config/file: check ExecStart for required files
# Fix application, then re-enable:
systemctl daemon-reload && systemctl start <service>
# Add StartLimitIntervalSec=0 to unit file to allow unlimited restarts during testing
```

Scenario 25: Audit Log Filling Up — Compliance Risk

In Simple Terms:

Audit logs record every important action for security and compliance. If the disk fills up with audit logs, the system may freeze (configured to be safe) or logs may be lost — both are compliance violations in financial services.

Technical Background:

Linux auditd writes to /var/log/audit/audit.log. When disk fills, auditd can be configured to suspend the system (SUSPEND), rotate logs, or halt. In financial environments, log retention is a regulatory requirement.

Diagnosis Commands:

```
df -h /var/log/audit   # Disk space
du -sh /var/log/audit/*   # Log file sizes
auditctl -s   # Auditd status
cat /etc/audit/auditd.conf | grep -E 'space_left|action|max_log'
# Check what's generating most events:
ausearch -ts today | aureport --file | head -20   # By file
ausearch -ts today | aureport --user | head -20   # By user
aureport --summary   # Overall summary
```

Reading the Output:

auditctl -s: 'backlog_limit=8192 backlog=8200' = audit backlog overflow (events being lost). /etc/audit/auditd.conf: 'space_left_action = SUSPEND' = system will freeze when low on space.

Resolution Steps:

```
# Immediate: free space
journalctl --vacuum-size=1G
find /var/log -name '*.gz' -mtime +30 -delete
# Increase log retention storage or add disk
lvextend -L +20G /dev/vg0/lv_audit && xfs_growfs /var/log/audit
# Tune auditd.conf for better rotation:
# max_log_file_action = ROTATE
# num_logs = 10
# max_log_file = 1024   (1GB per file)
# Configure log forwarding to SIEM (Splunk/Elastic):
# rsyslog: *.* @splunk_server:514
# Or audisp-remote plugin for auditd
```

```
service auditd restart   # Must use service, not systemctl for auditd
```

Scenario 26: Swap Usage Spiking — Memory Pressure

In Simple Terms:

Swap is like using a storage closet when your desk (RAM) is full. It works but is much slower — like walking to the closet and back for every document. Heavy swap use drastically slows down trading systems.

Technical Background:

Linux uses swap when RAM is under pressure. vm.swappiness controls swap aggressiveness. For latency-sensitive systems, swap should be disabled or heavily penalized. Page scanning and swapping add milliseconds of jitter.

Diagnosis Commands:

```
free -h   # RAM and swap overview
vmstat 1 10   # si/so = swap in/out per second
swapon -s   # Swap devices and usage
cat /proc/vmstat | grep -E 'pswpin|pswpout'   # Cumulative swap IO
smem -k -r | head -20   # Processes by memory with swap
cat /proc/<PID>/smaps_rollup | grep Swap   # Swap for specific process
sar -W 1 5   # Swap statistics over time
```

Reading the Output:

vmstat: si (swap in) and so (swap out) should be 0 in healthy system. Any non-zero value = active swapping. 'si=100 so=200' = severe memory pressure.

Resolution Steps:

```
# Identify top memory consumers:
ps aux --sort=-%mem | head -10
# If a process has a memory leak: restart it
# Tune swappiness (for production servers with enough RAM):
sysctl -w vm.swappiness=1   # Almost never swap (0 = never)
# Persistent: echo 'vm.swappiness=1' >> /etc/sysctl.conf
# Free page cache if RAM falsely appears full:
sync && echo 3 > /proc/sys/vm/drop_caches   # WARNING: performance impact
# For trading: disable swap entirely
swapoff -a   # Immediate (remove from /etc/fstab for permanent)
# Add RAM or tune application memory limits with cgroups
```

Scenario 27: Rsyslog Not Forwarding Logs to Central Server

In Simple Terms:

All server logs should be sent to a central collection point for security and compliance. If rsyslog stops forwarding logs, it's like a branch office that stops sending reports to headquarters — a compliance and security risk.

Technical Background:

Rsyslog forwards logs via UDP/TCP to remote syslog server. Failures can be caused by network issues, disk queue backlog, TLS certificate problems, or remote server capacity.

Diagnosis Commands:

```
systemctl status rsyslog   # Service status
rsyslogd -N1   # Validate config syntax
journalctl -u rsyslog -n 50   # Recent rsyslog logs
# Test connectivity to log server:
nc -zv <logserver> 514   # TCP syslog port
# Check queue:
ls -la /var/spool/rsyslog/   # Disk queue files
# Generate test message:
logger -p local0.info 'test message'   # Send test
# Check if received on log server:
tail -f /var/log/remote/<hostname>.log
```

Reading the Output:

rsyslogd -N1: Config errors shown. journalctl for rsyslog: 'action suspended' = cannot reach remote server. Queue files growing = backlog accumulating.

Resolution Steps:

```
# Fix network connectivity:
firewall-cmd --add-port=514/tcp --permanent && firewall-cmd --reload
# Fix config syntax errors from rsyslogd -N1
# If TLS cert issue: renew cert
# Clear stuck queue (warning: may lose queued messages):
systemctl stop rsyslog
rm /var/spool/rsyslog/*.qi 2>/dev/null
systemctl start rsyslog
# Tune queue settings in /etc/rsyslog.conf:
# $ActionQueueType LinkedList
# $ActionQueueFileName syslog_queue
# $ActionQueueMaxDiskSpace 1g
# $ActionQueueSaveOnShutdown on
systemctl restart rsyslog
```

Scenario 28: Cron Job Not Running

In Simple Terms:

Cron jobs are like scheduled tasks — run this script at 3 AM every day. When they silently stop running, scheduled backups, reports, or maintenance tasks are missed — often only discovered when something breaks.

Technical Background:

Cron failures can be caused by crond not running, wrong crontab syntax, environment variable differences between interactive and cron shells, or permission issues.

Diagnosis Commands:

```
systemctl status crond   # Is cron running?
crontab -l   # List cron jobs for current user
crontab -l -u <user>   # List for specific user
grep CRON /var/log/cron   # Cron execution logs
# Check system crontabs:
cat /etc/cron.d/*
ls /etc/cron.{hourly,daily,weekly,monthly}/
# Test cron environment (different from interactive):
env -i /bin/sh -c 'printenv'   # Minimal cron-like environment
```

Reading the Output:

/var/log/cron: 'CMD (/path/script.sh)' = ran. '/bin/sh: permission denied' = script not executable. 'command not found' = PATH issue in cron environment.

Resolution Steps:

```
# Fix common cron issues:
# 1. Script not executable:
chmod +x /path/to/script.sh
# 2. PATH issue: add full paths in crontab or set PATH
# In crontab: PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
# 3. Wrong crontab syntax (check online crontab validator)
# 4. Redirect output to catch errors:
# In crontab: 0 3 * * * /script.sh >> /var/log/script.log 2>&1
# 5. MAILTO for email output:
# In crontab: MAILTO=admin@company.com
# Restart crond if needed:
systemctl restart crond
# Use anacron for missed jobs:
cat /etc/anacrontab
```

Scenario 29: LVM Volume Group Out of Space — Cannot Extend

In Simple Terms:

LVM (Logical Volume Manager) is like a flexible filing cabinet system where you can resize drawers. But if the entire cabinet is full and you have no extra shelves, you need to add a new filing cabinet section.

Technical Background:

When a VG (Volume Group) has no free extents, LVM cannot extend existing LVs. Resolution requires adding a new PV (Physical Volume) or reducing existing LVs.

Diagnosis Commands:

```
pvs   # Physical volumes and free space
vgs   # Volume groups and free space
lvs   # Logical volumes
vgdisplay vg0   # Detailed VG info including free extents
lsblk   # Block device tree
fdisk -l   # Available disks
# Check if any LV can be reduced:
df -h   # Filesystem usage (if LV has lots of free space, can shrink)
```

Reading the Output:

vgs: 'VFree=0' = no free space in VG. pvs: Lists all PVs; if no new PVs visible, need new disk. lsblk: Shows unpartitioned disks that can be added.

Resolution Steps:

```
# Option 1: Add new disk to VG
# First partition new disk if needed:
parted /dev/sdb mklabel gpt
parted /dev/sdb mkpart primary 0% 100%
# Create PV:
pvcreate /dev/sdb1
# Add to VG:
vgextend vg0 /dev/sdb1
# Now extend LV:
lvextend -L +100G /dev/vg0/lv_data
xfs_growfs /data   # Grow XFS filesystem online
# resize2fs /dev/vg0/lv_data   # For ext4
# Option 2: Online PV resizing if underlying disk was extended:
pvresize /dev/sdb
```

Scenario 30: Multipath Storage Showing Degraded Path

In Simple Terms:

Enterprise storage is connected via multiple paths (like multiple roads to the same city) for redundancy. When a path fails, data still flows but you've lost your backup road. We need to identify and repair the failed path before both fail.

Technical Background:

Multipath I/O (MPIO) uses dm-multipath to present multiple physical paths to the same storage device as a single logical device. Path failures are caused by HBA issues, cable failures, switch faults, or storage controller problems.

Diagnosis Commands:

```
multipath -l   # List multipath devices and path status
multipath -ll   # Verbose with device details
multipathd show paths   # Path status in multipathd
multipathd show maps   # Map status
cat /sys/block/dm-X/dm/state   # Device state
# Check HBA status:
systool -c fc_host -v   # Fibre Channel host adapters
cat /sys/class/fc_host/host*/port_state   # FC port states
```

Reading the Output:

multipath -ll: '[failed faulty]' next to a path = path failed. '[active ready]' = healthy. '2:0:0:0 sdb 8:16 failed faulty' = sdb path down. FC port state: 'Linkdown' = FC link failure.

Resolution Steps:

```
# Remove failed path from multipath:
multipathd -k 'fail path sdb'
# Reinstate path after fixing physical issue:
multipathd -k 'reinstate path sdb'
# Reload multipath config:
multipath -r
# Restart multipathd:
systemctl restart multipathd
# Fix physical: check FC cable, HBA, switch zoning
# Rescan after fix:
echo '- - -' > /sys/class/scsi_host/host0/scan
# Verify:
multipath -l   # Should show [active ready] on all paths
# Check I/O is balanced:
iostat -x 1   # Both paths should show similar IO
```

SECTION 3: MASTER TROUBLESHOOTING FLOWCHART

This flowchart provides a systematic, step-by-step methodology to diagnose any production Linux issue. Follow each branch based on what you observe.

▶ STEP 1: INITIAL TRIAGE (First 60 seconds)

```
uptime                          # Load average (1min/5min/15min)
w                               # Who is logged in, what are they doing
date                            # Verify system time (NTP sync issues?)
last reboot                     # When did it last reboot?
dmesg -T | tail -30             # Recent kernel messages
```

✓ *Is load average > number of CPUs? → Go to STEP 2A. Are there kernel errors? → Go to STEP 2D. Is it a network issue? → Go to STEP 2C.*

▶ STEP 2A: HIGH LOAD AVERAGE INVESTIGATION

```
vmstat 1 5                      # r=runqueue b=blocked wa=iowait
# IF r > nCPUs: CPU bound → STEP 3A
# IF wa > 20: IO bound → STEP 3B
# IF b > 5: blocked processes → STEP 3C
mpstat -P ALL 1 5               # Per-CPU breakdown
top -b -n2 -d1 | head -40       # Which processes dominate CPU
```

✓ *Check vmstat columns: 'r' (run queue), 'b' (blocked), 'wa' (IO wait), 'si/so' (swap)*

▶ STEP 3A: CPU BOUND — FIND THE CULPRIT

```
ps -eo pid,ppid,%cpu,cmd --sort=-%cpu | head -10
pidstat -u 1 5                  # Per-process CPU over time
perf top -p <PID>               # Real-time CPU profiling
cat /proc/<PID>/wchan           # What kernel function is it in?
strace -cp <PID> sleep 5        # Syscall summary
# Check CPU frequency (throttling?):
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq
```

✓ *Is it a known application (DPDK busy polling = normal)? Is CPU frequency throttled? Is it a runaway loop?*

▶ STEP 3B: IO WAIT — IDENTIFY DISK BOTTLENECK

```
iostat -x 1 5                   # %util, await, svctm per device
# %util > 90 = device saturated
# await > 10ms = slow IO
iotop -a -o                     # Processes doing IO (only active)
lsof +D /mount/point            # What files are being accessed
blktrace -d /dev/sdb -a issue,complete -w 10 | blkparse -i -
# Check for NFS (could be remote IO wait):
nfsstat -c                      # NFS client stats
```

✓ *Single device 100% utilized? → storage bottleneck. Multiple devices? → application logic. NFS? → network/NFS server issue.*

▶ STEP 3C: PROCESSES BLOCKED IN D STATE

```
ps aux | grep ' D '             # Find D state processes
cat /proc/<PID>/wchan           # Kernel function waiting on
cat /proc/<PID>/stack           # Full kernel stack trace
# NFS check:
```

```
nfsstat -m && rpcinfo -p <nfs_server>
# Disk check:
dmesg | grep -i 'I/O error\|sd'
smartctl -a /dev/sdb            # Disk SMART health
```

✓ *wchan shows 'nfs_*'? → NFS issue. Shows 'jbd2_*'? → Journal flush hang. dmesg shows I/O errors? → Failing disk.*

▶ STEP 2B: MEMORY INVESTIGATION

```
free -h                         # RAM/swap overview
cat /proc/meminfo               # Detailed: MemAvailable, Dirty, Writeback
vmstat -s | head -20            # Memory statistics summary
smem -r -k | head -10          # Top memory consumers with swap
dmesg | grep -i oom             # OOM killer events
# Check memory overcommit:
cat /proc/sys/vm/overcommit_memory
```

✓ *OOM events? → identify and fix memory hog. Swap heavily used? → add RAM or tune swappiness. Dirty pages high? → IO bottleneck preventing writes.*

▶ STEP 2C: NETWORK INVESTIGATION

```
ip -s link show                 # TX/RX packets, errors, drops
ethtool -S eth0 | grep -i 'drop\|error\|missed'
ss -tnp                         # Active TCP connections
netstat -s | grep -E 'retransmit|failed|error'
# Latency check:
ping -c 100 <target> | tail -5  # Packet loss and latency
traceroute <target>             # Path and per-hop latency
# Connection tracking:
conntrack -L | wc -l            # Number of tracked connections
cat /proc/sys/net/netfilter/nf_conntrack_count
```

✓ *RX drops in ethtool? → ring buffer too small. High retransmits? → packet loss on path. Conntrack near limit? → connection table overflow.*

▶ STEP 2D: KERNEL/SYSTEM ERRORS

```
dmesg -T -l err,crit,alert,emerg   # Error-level kernel messages
journalctl -p err -b               # Systemd journal errors this boot
ausearch -m avc -ts today          # SELinux denials today
# Hardware errors:
mcelog --client                    # Machine check errors (CPU/memory)
edac-util -s 1                     # Memory ECC errors
smartctl -a /dev/sda               # Disk SMART status
ipmitool sel list | tail -20       # IPMI system event log
```

✓ *mcelog errors? → hardware failure (CPU/RAM). SMART errors? → disk failure imminent. SELinux AVC? → policy issue, not a real failure. IPMI SEL? → thermal/power events.*

▶ STEP 4: APPLICATION-LEVEL DIAGNOSIS

```
systemctl status <service>      # Service state and recent log
journalctl -u <service> -n 100  # Last 100 service log lines
# Process inspection:
ls -la /proc/<PID>/fd | wc -l   # File descriptor count
cat /proc/<PID>/limits          # Resource limits
pmap -x <PID> | tail -5         # Memory map summary
# Network connections of process:
ss -tnp | grep <PID>
```

```
# System calls:
strace -p <PID> -c sleep 10      # Syscall frequency summary
```

✓ *FDs near limit? → increase ulimits. Memory growing? → memory leak. Lots of syscalls? → application logic issue.*

▶ STEP 5: LOW-LATENCY SPECIFIC CHECKS

```
# CPU isolation and frequency:
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
cat /proc/cmdline | grep -o 'isolcpus=[^ ]*'
# IRQ affinity:
cat /proc/interrupts | grep eth
cat /proc/irq/*/smp_affinity_list
# NUMA check:
numastat -p <PID>                # NUMA memory locality for process
numactl --hardware               # NUMA topology
# OS jitter measurement:
cyclictest -p99 -t1 -m -i 200 -D 30   # Measure scheduling latency
# SMI count:
rdmsr 0x34                       # Read SMI counter (requires msr module)
```

✓ *cyclictest max > 100us? → OS jitter problem (IRQs, power management). SMI count increasing? → firmware SMI causing jitter. NUMA imbalance? → cross-NUMA memory access.*

▶ STEP 6: RESOLUTION AND DOCUMENTATION

```
# Before making changes — capture baseline:
date; hostname; uname -r; uptime
vmstat 1 1; free -h; df -h
ip -s link show; ethtool -S eth0 | grep -i drop
# After fix — verify improvement:
# Compare same metrics as baseline
# Document in change management system:
# - Root cause identified
# - Actions taken with exact commands
# - Result verification
# - Prevention measures added
# Check monitoring shows recovery:
# Prometheus/Grafana dashboard should normalize
```

✓ *Is the issue fully resolved? Have you verified with the application/monitoring team? Have you documented the RCA and prevention steps?*

QUICK REFERENCE: KEY COMMANDS BY CATEGORY

Category	Diagnosis Commands	What to Check
CPU	`top, htop, mpstat -P ALL 1, pidstat -u, perf top`	Load avg, per-CPU %, runqueue length, hot functions
Memory	`free -h, vmstat -s, smem -r, cat /proc/meminfo`	MemAvailable, swap usage, OOM events, page faults
Disk IO	`iostat -x 1, iotop, blktrace, smartctl -a`	%util, await, read/write IOPS, disk health
Network	`ip -s link, ethtool -S, ss -tnp, netstat -s`	TX/RX errors, drops, retransmits, open sockets
Processes	`ps aux, pstree, pidstat, strace, lsof -p`	State (D/Z/R/S), FD count, open files, syscalls
Latency	`cyclictest, perf stat, numastat, rdmsr 0x34`	Jitter, cache misses, NUMA locality, SMI count
Storage	`multipath -l, pvs/vgs/lvs, df -i`	Path status, VG free space, inode usage
Security	`ausearch -m avc, sestatus, iptables -L -nv`	SELinux denials, firewall rules, audit events
Services	`systemctl status, journalctl -u, coredumpctl`	Service state, crash logs, core dumps
Kernel	`dmesg -T, /proc/sys/*, sysctl -a`	Hardware errors, kernel params, module issues

Good luck with your interview at Morgan Stanley Montreal! — Prepared for Director-Level Infrastructure SRE Role