T1 Exercises – Mehran Amiri

1. Please summarize the following server components:

Motherboard (System Board): The main circuit board connecting all components (CPU, RAM, storage, etc.), providing communication pathways via buses and chipsets. It determines compatibility and expansion capabilities.

CPU (Central Processing Unit): The server's "brain," executing instructions and processing data. Server CPUs (e.g., Intel Xeon, AMD EPYC) are optimized for multi-core performance, handling high workloads and virtualization.

RAM (Memory): Temporary, high-speed storage for active data and applications. Servers typically use ECC (Error-Correcting Code) RAM to ensure data integrity, with capacities ranging from tens to hundreds of GBs for multitasking and virtualization.

Storage Drives (HDD, SSD/NVMe): Persistent storage for data and the OS. HDDs offer high capacity for bulk storage, SSDs provide faster access for databases, and NVMe SSDs deliver ultra-low latency for high-performance applications.

RAID Controller (Smart Array): Manages multiple storage drives in a RAID array for redundancy (e.g., RAID 1 mirroring) or performance (e.g., RAID 0 striping). Smart Array controllers (e.g., HPE's) often include caching and battery backup for data protection.

Power Supply Unit (PSU): Converts AC power to DC, supplying electricity to all components. Server PSUs are often redundant (dual PSUs) and hot-swappable to ensure uptime, with high efficiency (e.g., 80 PLUS Platinum) to reduce energy costs.

Network Interface Card (NIC): Enables network connectivity, handling data transfer between the server and external networks. Server NICs support high speeds (e.g., 10/25/100 Gbps) and features like RDMA for low-latency communication.

Cooling System (Fans and Heat Sinks): Maintains optimal temperatures by dissipating heat from components like the CPU and GPU. Servers use advanced cooling (e.g., multiple fans, liquid cooling in high-end systems) to prevent thermal throttling and ensure reliability.

Expansion Slots (PCIe): Slots on the motherboard (e.g., PCIe x16) for adding cards like GPUs, additional NICs, or storage controllers. PCIe slots in servers support high bandwidth for accelerators and I/O expansion.

Chassis (Rack/Tower/Blade): The physical enclosure housing components. Rack chassis (e.g., 1U, 2U) are space-efficient for data centers, towers suit smaller setups, and blade servers maximize density in blade enclosures.

BIOS/UEFI Firmware: Low-level software initializing hardware during boot and providing the interface between hardware and OS. UEFI (modern replacement for BIOS) supports advanced features like Secure Boot and larger disks.

Backplane: A circuit board connecting drives to the RAID controller or motherboard, often supporting hot-swapping. In blade servers, it connects multiple blades to shared power and networking resources.

System Battery: Typically a CMOS battery on the motherboard, maintaining BIOS/UEFI settings and the system clock when powered off. Some servers also have battery backup units (BBUs) for RAID controllers to protect cached data during power loss.

2. What are IPMI and iLO, and what are their functions?

IPMI(Intelligent Platform Management Interface) or iLO (Integrated Lights-Out)are management interfaces that you can use to operate the server remotely. They enable operation as if you were directly on the server.

They allow system administrators to monitor and control servers even when the operating system is not running or the server is powered off.

They also provide information about the installed hardware and its status.

iLO, , is a proprietary implementation similar to IPMI but with advanced features, including a graphical web interface, virtual media support, and remote console access. It offers enhanced usability and security over standard IPMI.

3. How do IPMI or iLO relate to the BIOS or UEFI firmware?

IPMI and iLO do not replace or directly integrate into the BIOS/UEFI firmware, but they interact with it in meaningful ways. Both technologies operate independently from the host operating system and provide remote access to low-level hardware functions, including those managed by BIOS/UEFI.

Key Interactions:

- **Remote BIOS Access:**
  Using iLO or IPMI with KVM capabilities, administrators can access and interact with the BIOS or UEFI interface remotely—even before the OS boots.

- **Hardware Initialization Feedback:**
  iLO/IPMI can read and report values such as fan speeds, temperatures, and power statuses that are initially configured by the BIOS.

- **Boot Sequence Control:**
  Some iLO implementations allow remote boot order changes, although the changes are ultimately handled by the BIOS/UEFI.

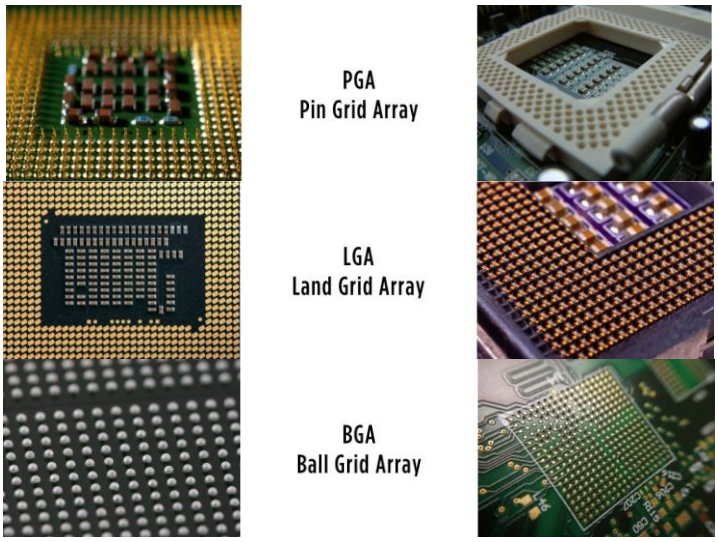
- **Event Logging:**
  IPMI logs may include events related to BIOS, such as failed POST (Power-On Self Test), hardware initialization errors, or firmware changes.

4. What are CPU sockets on a server, and what is their purpose?

A CPU socket is a specific part on a motherboard that is purposely designed to hold a central processing unit (CPU). A CPU socket or CPU slot is designed with thousands of pins or contact points for power and data transfer between the CPU and the rest of the processors on the motherboard.
In servers, the CPU socket plays a crucial role in determining the system's scalability, performance, and upgradeability.

Key Characteristics:

1. **Number of Sockets:**
   - **Single-socket servers** support one CPU and are suitable for entry-level or lightweight workloads.
   - **Multi-socket servers** (e.g., dual-, quad-, or even octa-socket) support multiple physical CPUs, significantly boosting processing power and memory capacity.
2. **Socket Type:**
   - Defined by physical layout, pin configuration, and electrical compatibility.



3. **Upgrade and Compatibility:**
    - The socket type determines which CPU models and generations are supported.
    - Upgrading CPUs often requires the same socket and chipset family.

5. Why was the pseudo file system introduced in Linux?

The pseudo file system in Linux, such as /proc and /sys, was introduced to align with the Unix philosophy of:

"Everything is a file"

These virtual file systems provide a consistent and simple interface for accessing kernel and hardware information using standard file operations (read/write), without relying on complex APIs or tools.

Here are the key reasons for its introduction:

1. Dynamic Information Access

Real-Time Data: The pseudo file system allows users and applications to access real-time information about the system's state, including process information, system statistics, and kernel parameters. This data is generated dynamically by the kernel, reflecting the current state of the system without needing to store it on disk.

2. Simplified Interface

File-Based Access: By presenting system information as files and directories, the pseudo file system provides a familiar and straightforward interface for users and applications. Users can read from and write to these files using standard file operations, making it easy to interact with the kernel and retrieve system information.

3. Process Management

Process Information: The /proc file system contains directories for each running process, identified by their process IDs (PIDs). Within these directories, files provide detailed information about each process, such as memory usage, CPU time, and open file descriptors. This makes it easier for system administrators and developers to monitor and manage processes.

4. Kernel Configuration and Tuning

Kernel Parameters: The pseudo file system allows users to modify certain kernel parameters at runtime through files in /proc,/sys. This enables dynamic tuning of system behavior without requiring a reboot, facilitating performance optimization and configuration changes on the fly.

5. Resource Monitoring

System Statistics: The /proc file system provides access to various system statistics, such as CPU usage, memory usage, disk I/O, and network activity. This information is crucial for monitoring system performance and diagnosing issues.

6. Separation from Physical Storage

No Disk Space Usage: Since the pseudo file system does not correspond to actual files stored on disk, it does not consume disk space. The data is generated in memory, which helps keep the system efficient and responsive.

7. Unified View of System Resources

Centralized Information: The pseudo file system provides a centralized location for accessing a wide range of system information, making it easier for users and applications to gather data about the system's state and performance.

6. What are the differences between a pseudo file system and a normal file system?

1. Purpose and Functionality

- **Pseudo File System**:

- Designed to provide a dynamic interface to kernel and system information. It exposes real-time data about system processes, hardware, and kernel parameters.
- Examples include /proc and /sys in Linux, which allow users to access information about running processes, system statistics, and kernel settings.

- **Normal File System**:
  - Used for storing and organizing files on physical storage devices (like hard drives, SSDs, etc.). It manages how data is stored, retrieved, and organized on disk.
  - Examples include ext4, NTFS, FAT32, and others, which are used to manage user data and application files.

2. Data Storage

- **Pseudo File System**:
  - Does not store data on disk. Instead, it generates data dynamically in memory when accessed. The contents of a pseudo file system can change frequently based on the current state of the system.
  - For example, reading from /proc/cpuinfo provides real-time information about the CPU, which is generated by the kernel at the moment of access.
- **Normal File System**:
  - Stores data persistently on physical storage media. The data remains intact even when the system is powered off, and it is organized into files and directories.
  - For example, files stored in a normal file system can include documents, images, applications, and system files.

3. File Operations

- **Pseudo File System**:
  - Operations on files in a pseudo file system often reflect real-time changes and may not involve traditional file I/O operations. For instance, writing to certain files in /proc can change kernel parameters or system behavior immediately.

        - The files may not have a fixed size or content, as they are generated on-the-fly.

- **Normal File System**:

    - Supports standard file operations such as reading, writing, deleting, and modifying files. The files have a defined structure, size, and content that persists until explicitly changed or deleted.

    - File operations are typically managed by the file system's metadata, which keeps track of file locations, sizes, and permissions.

4. Structure and Hierarchy

- **Pseudo File System**:

    - The structure is often hierarchical but is designed to represent system information rather than user data. The organization of files and directories reflects the system's architecture and operational parameters.

    - For example, /proc contains directories for each process, as well as files for system statistics.

- **Normal File System**:

    - The structure is organized around user data and applications, with a focus on usability and accessibility for end-users. It includes directories and files that users create and manage.

    - For example, a typical user directory structure might include folders for documents, downloads, and media.

5. Persistence

- **Pseudo File System**:

    - Data is transient and does not persist across reboots. Once the system is restarted, the contents of the pseudo file system are regenerated based on the current state of the system.

- **Normal File System**:

    - Data is persistent and remains available across reboots. Files and directories are stored on disk and can be accessed at any time until they are deleted or modified.

7. What kind of information is available in the /sys/ directory?

The /sys/ directory in Linux is part of the sysfs virtual file system, which provides a structured interface to kernel objects and their attributes. It exposes information about various aspects of the kernel, devices, and system configuration. Here are some key types of information available in the /sys/ directory:

1. Device Information

- **Device Drivers**: Information about device drivers loaded in the kernel, including their status and parameters.
- **Device Attributes**: Each device has a directory under /sys/class/ or /sys/bus/ that contains attributes such as device status, configuration options, and capabilities.

2. Kernel Parameters

- **Kernel Configuration**: Parameters that can be read or modified to change kernel behavior. These are often found under /sys/module/ and /sys/kernel/.
- **Dynamic Tuning**: Certain kernel parameters can be adjusted at runtime, allowing for dynamic tuning of system performance and behavior.

3. System Information

- **CPU Information**: Details about the CPU(s) in the system, including core counts, CPU frequency, and topology. This information can be found under /sys/devices/system/cpu/.
- **Memory Information**: Information about system memory, including available memory, memory zones, and memory usage statistics.

4. Power Management

- **Power States**: Information about power management features, including CPU power states and device power management settings. This can be found under /sys/class/power_supply/ and /sys/devices/system/cpu/cpu*/cpuidle/.

5. Block Devices

- **Storage Devices**: Information about block devices (like hard drives and SSDs) is available under /sys/block/. This includes details about partitions, device attributes, and performance statistics.

6. Network Interfaces

- **Network Device Information**: Details about network interfaces, including their status, configuration, and statistics. This information can be found under /sys/class/net/.

7. File System Information

- **Mount Points**: Information about mounted file systems and their attributes can be found under /sys/fs/.

8. Thermal Management

- **Thermal Zones**: Information about thermal zones and temperature sensors, which can be found under /sys/class/thermal/. This includes temperature readings and cooling device information.

9. Firmware Loading:

The /sys/firmware/ directory contains information related to firmware interfaces and loaded firmware. This can include details about firmware for various devices, such as network cards, graphics cards, and other hardware components that require firmware to operate.

For example, you might find subdirectories or files related to specific firmware interfaces, such as ACPI (Advanced Configuration and Power Interface) or EFI (Extensible Firmware Interface).

- **ACPI Information:**

  The /sys/firmware/acpi/ directory provides information about the ACPI subsystem, which is responsible for power management and hardware configuration. It includes details about ACPI tables, devices, and power states.

- **EFI Information:**

  If your system uses UEFI, you may find information related to UEFI firmware in /sys/firmware/efi/. This can include variables, runtime services, and other UEFI-related data.

10. Device Hierarchy

- **Device Tree**: The /sys/ directory reflects the hierarchy of devices in the system, showing how devices are connected and their relationships. This is useful for understanding the system's hardware layout.

8. What is DMA (Direct Memory Access), and what is its use case in Linux?

Direct Memory Access (DMA) is a hardware mechanism that enables peripherals to transfer data to/from memory without CPU involvement, managed by a DMA controller. In Linux, DMA is used in storage (e.g., SSDs), networking (e.g., NICs), graphics, and embedded systems to optimize high-throughput data transfers. The Linux kernel provides a DMA API, supports scatter-gather DMA, and uses IOMMU for security. DMA reduces CPU load, boosts throughput, and enhances scalability.

9. What does the lsblk command do internally when executed in Linux?

lsblk (List Block Devices)

Displays information about block devices (e.g., disks, partitions, LVM volumes) in a tree-like format. Relies on /sys/block, and /proc/partitions.

Output: Device names, sizes, file system types, mount points

lspci (List PCI Devices)

Displays information about PCI/PCIe hardware connected to the system (e.g., GPUs, NICs, USB controllers)., such as network cards, graphics cards, and storage controllers.

Reads data from /sys/bus/pci/devices/ or /proc/bus/pci/ (via libpci or direct kernel interfaces).

Output: PCI device list with vendor, device name, and bus address.

lshw (List Hardware)

Provides a detailed overview of all hardware components in the system, including CPU, memory, disks, PCI devices, USB devices, and more.

Gathers data from multiple sources:

/sys/ (sysfs) for device information.

/proc/ (e.g., /proc/cpuinfo, /proc/meminfo) for CPU and memory details.

Can output in various formats (text, XML, JSON).

Output: A hierarchical view of hardware, including block devices, PCI devices, and more.

10. How can we simulate a shutdown operation via the /sys file system?

To simulate a shutdown, write disk to /sys/power/state, which initiates a hibernate-like operation (saving system state to disk and powering off):

```
echo disk > /sys/power/state
```

Alternatively, use /sys/power/disk to configure the shutdown mode before triggering:

```
echo shutdown > /sys/power/disk

echo disk > /sys/power/state
```

This tells the system to prepare for a shutdown-like state (hibernation).

It is not supported in VirtualBox!!

```
mehran@mehran-VirtualBox:~$ sudo sh -c 'echo test > /sys/power/disk'
sh: 1: echo: echo: I/O error
mehran@mehran-VirtualBox:~$ sudo sh -c 'echo disk > /sys/power/state'
sh: 1: echo: echo: I/O error
```

11. What are the different types of kernels?

Kernels, the core of operating systems, manage hardware, processes, and resources. The main types are:

- **Monolithic Kernel**: A single, comprehensive kernel handling all tasks (e.g., process management, drivers) in kernel space, offering high performance but lower modularity. Example: Linux (with loadable modules).
- **Microkernel**: A minimal kernel managing only essential functions (e.g., IPC, scheduling), with drivers and services in user space, enhancing reliability and modularity but with performance overhead. Example: QNX, Minix.
- **Hybrid Kernel**: Combines monolithic performance with microkernel modularity, running critical services in kernel space and others in user space. Example: Windows NT, macOS (XNU).
- **Exokernel**: An experimental design minimizing abstractions, allowing applications direct hardware access for customized performance, but complex to develop. Example: MIT Exokernel.
- **Nanokernel**: An ultra-minimal kernel providing basic hardware abstraction, used in secure or real-time systems, with most functionality in user space. Example: seL4.

12. Why is the first sector of a disk used for the MBR (Master Boot Record)?

The **Master Boot Record (MBR)** is stored in the **first sector** (sector 0, 512 bytes) of a disk because it's a fixed, accessible location where the BIOS looks to load the bootloader and partition table, initiating the boot process. This placement, established by early IBM PC standards, ensures simplicity, compatibility, and efficiency. The MBR contains bootloader code (446 bytes), a partition table (64 bytes), and a boot signature (2 bytes). Its role is critical for bootstrapping the OS and defining disk partitions, but it's limited by size and tied to legacy BIOS systems, with modern systems shifting to UEFI and GPT.

Standardized Boot Process: Ensures a predictable, universal location for BIOS to find boot code, enabling consistent booting across systems.

Fast Boot Initiation: Immediate access to the bootloader minimizes boot time.

Efficient Disk Use: Fits essential boot and partition data in a single sector, optimizing space.

Legacy Compatibility: Supports older BIOS-based systems and dual-boot setups, maintaining interoperability.

Critical for Partitioning: Defines primary partitions, organizing the disk for OS access.

Single Point of Failure: Its pivotal role means corruption can halt booting, underscoring its significance in system reliability.

13. If the MBR is located in the first 512 bytes, how does it know the location of GRUB or another bootloader to load the kernel?

Limited MBR Role: The MBR's bootloader is minimal and only needs to know the disk sectors of the next stage (e.g., GRUB Stage 1.5 or VBR). It doesn't directly locate the kernel or GRUB's main files.

GRUB Installation: Tools like grub-install configure the MBR bootloader with the necessary sector addresses, making the process seamless.

Partition Table Dependency: The MBR relies on the partition table to identify the active partition's boot sector, but GRUB often bypasses this by using the post-MBR gap for Stage 1.5.

File System Independence: The MBR bootloader doesn't understand file systems; it uses raw sector addresses. GRUB Stage 1.5 or later adds file system support to locate configuration and kernel files

14. What are `.efi` files, and what is their role in the boot process?

.efi files are executable files used in the Unified Extensible Firmware Interface (UEFI) boot process. They are part of the UEFI firmware standard, which has largely replaced the legacy BIOS for booting modern computers. These files contain machine code in the Portable Executable (PE) format, designed to run in the UEFI environment before the operating system is loaded.

The .efi files are UEFI applications, drivers, or bootloaders compiled to run on UEFI firmware. They are typically written in C or assembly and follow the UEFI specification.

.efi files perform tasks such as initializing hardware, loading drivers, or acting as bootloaders to start the operating system

15. What is the ESP (EFI System Partition), and how is it used in UEFI?

The EFI System Partition (ESP) is a FAT-formatted partition (100–512 MB) on UEFI systems, storing .efi bootloaders (e.g., GRUB, Windows Boot Manager), UEFI applications, and drivers. It is identified by a specific GUID and used by the UEFI firmware to locate and execute .efi files, which load the OS kernel or run pre-boot utilities. The ESP supports multi-boot, Secure Boot, and configuration storage, typically mounted at /boot/efi in Linux.

16. Please explain the following section from `/etc/grub/grub.conf`

```
menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os
$menuentry_id_option 'gnulinux-simple-3e3d2181-a1f5-4456-867c-a69f52c910e6'
{
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
        insmod part_gpt
        insmod ext2
        set root='hd0,gpt2'
```

```
      if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2
--hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2
49ee8c4e-7d13-455b-b287-488a33286e30
      else
        search --no-floppy --fs-uuid --set=root
49ee8c4e-7d13-455b-b287-488a33286e30
      fi
      linux /vmlinuz-5.4.0-65-generic root=/dev/mapper/vg0-root ro
maybe-ubiquity
      initrd      /initrd.img-5.4.0-65-generic
}
```

recordfail

- A GRUB command that checks if the previous boot failed (e.g., due to a timeout or crash). If a failure is recorded, GRUB may adjust its behavior, such as showing the menu instead of auto-booting, to allow user intervention.

load_video

- Loads video drivers to initialize graphical capabilities for the GRUB menu (e.g., to display a graphical boot menu or splash screen). This ensures GRUB can use the framebuffer or other video modes.

gfxmode $linux_gfx_mode

- Sets the graphical resolution for GRUB's display, using the value of the variable $linux_gfx_mode (defined elsewhere, often in /etc/default/grub, e.g., GRUB_GFXMODE=1024x768). This controls the screen resolution during the GRUB menu display.

insmod gzio

- Loads the gzio module, which enables GRUB to handle gzip-compressed files. This is often needed to read compressed kernel images (e.g., vmlinuz) or initramfs files.

if [x$grub_platform = xxen]; then insmod xzio; insmod lzopio; fi

- A conditional statement checking if GRUB is running on the Xen hypervisor ($grub_platform = xen).
    - If true, it loads the xzio and lzopio modules, which provide support for Xen-specific compression formats (e.g., for Xen domU kernels).

- This ensures compatibility when booting a Linux kernel in a Xen virtualized environment.

insmod part_gpt

- Loads the part_gpt module, enabling GRUB to understand and navigate GPT (GUID Partition Table) disk layouts. This is necessary for systems using GPT instead of the older MBR partitioning scheme.

insmod ext2

- Loads the ext2 module, allowing GRUB to read ext2/ext3/ext4 file systems (GRUB treats ext3/ext4 as ext2 for basic operations). This is needed to access the /boot partition where the kernel and initramfs reside.

set root='hd0,gpt2'

- Sets the root device for GRUB, specifying where to find the boot files (kernel, initramfs, etc.).
    - hd0: Refers to the first disk (GRUB numbering starts at 0).
    - gpt2: Refers to the second partition on a GPT-partitioned disk.
    - This tells GRUB to look for boot files on the second partition of the first disk.

Use for Ensures GRUB can reliably find the root partition even if disk order changes (e.g., adding/removing drives), using UUID for robustness.

if [x$feature_platform_search_hint = xy]:

Checks if GRUB supports search hints (a performance optimization feature).

--no-floppy: Skips searching floppy drives (historical option, rarely relevant).

--fs-uuid: Searches for a partition with the specified UUID.

--set=root: Sets the found partition as GRUB's root.

--hint-*: Provides hints for faster lookup:

--hint-bios=hd0,gpt2: Hint for BIOS systems.

--hint-efi=hd0,gpt2: Hint for UEFI systems.

--hint-baremetal=ahci0,gpt2: Hint for bare-metal AHCI controllers

Loads the kernel into memory with the necessary parameters to boot the OS.

/vmlinuz-5.4.0-65-generic: Path to the kernel image (relative to the root partition set earlier, /boot/vmlinuz-...).

root=/dev/mapper/vg0-root: Specifies the root file system for the OS, here using a logical volume (vg0-root) managed by LVM.

ro: Mounts the root file system as read-only during initial boot (later remounted as read-write).

maybe-ubiquity: A Ubuntu-specific parameter, often used during live CD/installer boots to enable the Ubiquity installer if present.

Overall of this Section

This section defines a GRUB 2 boot entry for an Ubuntu system:

- It sets up the environment by loading necessary modules (gzio, part_gpt, ext2, etc.) to read GPT partitions and ext2/3/4 file systems.
- It locates the root partition using a UUID, ensuring reliability across disk changes.
- It loads the Linux kernel (vmlinuz-5.4.0-65-generic) and initramfs (initrd.img-5.4.0-65-generic) with parameters to boot Ubuntu from an LVM-managed root file system.