به نام خدا

1. In fdisk, when using the p (print) command, there is a column labeled "sectors." What does "sector" refer to in this context?
In computer disk storage, a **sector** is a subdivision of a track on a magnetic disk or optical disc. For most disks, each sector stores a fixed amount of user-accessible data, traditionally 512 bytes for hard disk drives (HDDs), and 2048 bytes for CD-ROMs, DVD-ROMs and BD-ROMs. Newer HDDs and SSDs use 4096 byte (4 KiB) sectors, which are known as the Advanced Format (AF).
In fdisk, the "sectors" column refers to the size of the partition in 512-byte sectors.

2. There are three types of servers mentioned. Please summarize your partitioning recommendations for each:

● Linux desktop systems

- **/ (root)**: 30–50GB (system files, apps).
- **/home**: Remaining space (user data, personal files).
- **swap**: Equal to RAM (or 2x if hibernation needed).
- **/boot**: (500MB–1GB, UEFI/legacy boot).
- **/var**: (5–10GB if running local services).

● Linux servers used for databases or web services with extensive logging

- **/ (root)**: 10–20GB (minimal OS footprint).
- **/var**: 50GB+ (logs, databases, web content).
- **/home**: Minimal (if any; servers often lack user storage).
- **/opt** or **/srv**: As needed (application data).
- swap: 1.5x RAM (for database tuning).

● Linux servers used in university labs or staging environments where multiple users require individual home directories and personal storage space

- **/ (root)**: 30–40GB (OS + core software).
- **/home**: Large (50%+ of disk; user projects/configs).
- **/tmp**: 5–10GB (scratch space, tmpfs for speed).
- swap: Equal to RAM (or 1.5x for lab workloads).

- Optional:
    - Separate **/opt** (10–20GB for lab software).
    - Automated cleanup for **/tmp** and **/var/tmp**.

3. How does an operating system run multiple applications when the total available RAM is less than the combined memory requirements?

The OS uses virtual memory and paging to run multiple apps with limited RAM:

1. Memory Overcommitment – Allows apps to request more memory than physically available (RAM + swap).
2. Demand Paging – Loads only actively used parts of apps into RAM (lazy loading).
3. Swapping – Moves inactive pages to disk (swap space) when RAM is full.
4. Page Replacement – Algorithms (e.g., LRU) evict least-used pages to free RAM.

Please summarize:

● How processes are scheduled to run on the CPU

The OS uses a **scheduler** to allocate CPU time to processes, ensuring fairness, efficiency, and responsiveness. Here's how it works:

1. **Ready Queue** – Processes ready to run are placed in a queue.
2. **Scheduling Algorithm** – The scheduler picks the next process based on a policy:
    - **Preemptive** (e.g., Round Robin, Linux CFS) – Forces running processes to yield CPU after a time slice.
    - **Non-Preemptive** (e.g., FIFO) – Processes run until they block or finish.
3. **Context Switch** – The OS saves the current process's state and loads the next one.
4. **Execution** – The selected process runs on the CPU until it:
    - Completes,
    - Blocks (e.g., I/O wait), or
    - Gets preempted (time slice expires).

● How process data is loaded into RAM

When a process starts, the OS loads its data into RAM in stages:

1. Program Loading – The executable file (e.g., .exe, .elf) is read from disk.

2. Memory Allocation – The OS allocates:
    - Code (Text Segment) → Executable instructions.
    - Data Segment → Global/static variables.
    - Heap → Dynamically allocated memory (malloc, new).
    - Stack → Local variables, function calls.
3. Demand Paging – Instead of loading everything at once:
    - Only necessary pages are loaded initially.
    - Other pages are fetched from disk on-demand (when accessed).
4. Page Tables – The OS maps virtual addresses (used by the process) to physical RAM (or swap if RAM is full).
5. Dynamic Linking – Shared libraries (e.g., .dll, .so) are loaded when needed.

● What happens when RAM is insufficient for a new process

1. Swapping (Paging Out)

- The OS moves inactive pages (from idle processes) from RAM to swap space (disk).
- Frees up RAM for the new process.

2. OOM Killer (Out-of-Memory Killer)

- On Linux, if swapping isn't enough, the kernel forcibly terminates processes (often the largest one) to reclaim memory.

3. Thrashing (Severe Performance Drop)

- If RAM and swap are overloaded, the system spends more time swapping pages than running processes, causing extreme slowdowns.

4. Allocation Failure

- If no memory can be freed, the new process fails to start (e.g., malloc() returns NULL, apps crash with "Out of Memory" errors).

● How and when the operating system uses disk space (paging and swapping)

1. When RAM is Full
    - The OS moves less-used pages from RAM to disk (swap space) to free up memory.
2. On Demand Paging

    - Initially, only parts of a process are loaded into RAM; unused pages stay on disk until needed.
3. During Page Faults
    - If a process accesses a page not in RAM, the OS fetches it from disk (causing a slight delay).

How It Works

- Swapping (Whole Process)
    - Rare in modern systems; an entire idle process may be moved to disk.
- Paging (Individual Pages)
    - Only inactive 4KB (or 2MB with huge pages) chunks are swapped out.
- Page Replacement Algorithms
    - The OS selects pages to evict (e.g., LRU—Least Recently Used).

● The memory management strategies involved (e.g., demand paging)

1. Demand Paging

- What: Loads process pages into RAM only when accessed (triggered by page faults).
- Why: Saves memory by avoiding upfront loading of entire programs.
- Example: Starting a 2GB game only loads essential parts first (e.g., menus).

2. Page Replacement

- What: Decides which RAM pages to evict when space is needed (e.g., LRU, FIFO).
- Why: Keeps active processes in RAM while offloading idle ones to disk.

3. Swapping

- What: Moves entire idle processes (or pages) to disk when RAM is full.
- Why: Emergency measure to prevent crashes, but slows performance.

4. Memory Overcommitment

- What: Allows apps to reserve more memory than physically available (backed by swap).

- Why: Improves flexibility but risks OOM (Out-of-Memory) kills if overused.

5. Huge Pages

- What: Uses larger page sizes (2MB/1GB vs. 4KB) for big workloads.
- Why: Reduces TLB misses and overhead for databases/HPC.

6. Copy-on-Write (CoW)

- What: Shares memory between processes until one modifies it (then copies).
- Why: Saves RAM for duplicate data (e.g., fork() in Linux).

● Whether all pages of a process must be loaded into RAM for execution

No, not all pages of a process need to be loaded into RAM for execution. Modern operating systems use demand paging, a memory management strategy that loads pages only when they are needed (on-demand). Here's how it works:

4. What is the Translation Lookaside Buffer (TLB), and what role does it play in memory management?

A translation lookaside buffer (TLB) is a memory cache that stores the recent translations of virtual memory to physical memory. It is used to reduce the time taken to access a user memory location. It can be called an address-translation cache. It is a part of the chip's memory-management unit (MMU). A TLB may reside between the CPU and the CPU cache, between CPU cache and the main memory or between the different levels of the multi-level cache. The majority of desktop, laptop, and server processors include one or more TLBs in the memory-management hardware, and it is nearly always present in any processor that uses paged or segmented virtual memory.

The TLB is sometimes implemented as content-addressable memory (CAM). The CAM search key is the virtual address, and the search result is a physical address. If the requested address is present in the TLB, the CAM search yields a match quickly and the retrieved physical address can be used to access memory. This is called a TLB hit. If the requested address is not in the TLB, it is a miss, and the translation proceeds by looking up the page table in a process called a *page walk*. The page walk is time-consuming when compared to the processor speed, as it involves reading the contents of multiple memory locations and using them to compute the physical address. After the physical address is determined by the page walk, the virtual address to physical address mapping is entered into the TLB. The PowerPC 604, for example, has a two-way set-associative TLB for data loads and stores.Some processors have different instruction and data address TLBs.

5. What are a page, a virtual page, and a context switch?

Page: A fixed-size block of memory (e.g., 4 KB) used by the OS for managing physical RAM.

Virtual Page: A page in virtual memory, which may reside in RAM or on disk (swap).

Context Switch: When the OS saves the state of one process and restores another, including registers, memory mappings, and CPU state.

Additionally, how does increasing swap space affect context-switching performance?

- Increasing swap space can degrade performance if heavy swapping occurs (more disk I/O during context switches).
- If RAM is sufficient, extra swap has minimal impact, but excessive swapping (thrashing) slows down context switches due to disk latency.

How does page size influence these effects?

- **Larger pages** reduce TLB misses (faster context switches) but increase internal fragmentation and swap I/O overhead (slower when swapping).
- **Smaller pages** minimize wasted memory but raise TLB misses, potentially slowing context switches (unless TLB coverage is good).

6. What is a huge page? Please explain its purpose and when it is used.
A huge page is a memory page significantly larger than the standard page size (e.g., 2MB or 1GB instead of 4KB). It is used to improve system performance by reducing Translation Lookaside Buffer (TLB) misses and lowering page table overhead.

Purpose of Huge Pages

1. Reduce TLB Pressure
    - Larger pages mean fewer entries needed to cover the same memory range → fewer TLB misses → faster memory access.
2. Lower Page Table Overhead
    - Fewer pages → smaller page tables → less memory used for metadata.
3. Improve Performance for Large Workloads
    - Critical for databases (Oracle, PostgreSQL), HPC, and big-data apps that access large memory regions.

When Are Huge Pages Used?

- **Memory-Intensive Applications** (e.g., Redis, MySQL, JVM with -XX:+UseLargePages)
- **Virtualization & Cloud Computing** (KVM, VMware benefit from reduced hypervisor overhead)
- **High-Performance Computing (HPC)** (Scientific simulations, GPU workloads)
- **Real-Time Systems** (Predictable latency by minimizing TLB thrashing)

7. What is memory fragmentation in RAM, and what problems can it cause?

In main memory fragmentation, when a computer program requests blocks of memory from the computer system, the blocks are allocated in chunks. When the computer program is finished with a chunk, it can free it back to the system, making it available to later be allocated again to another or the same program. The size and the amount of time a chunk is held by a program varies. During its lifespan, a computer program can request and free many chunks of memory.

Fragmentation can occur when a block of memory is requested by a program, and is allocated to that program, but the program has not freed it. This leads to theoretically "available", unused memory, being marked as allocated - which reduces the amount of globally available memory, making it harder for programs to request and access memory.

When a program is started, the free memory areas are long and contiguous. Over time and with use, the long contiguous regions become fragmented into smaller and smaller contiguous areas. Eventually, it may become impossible for the program to obtain large contiguous chunks of memory.