**Memory Management [ free, vmstat, swapon, swapoff, mkswap, iostat, sar, sync, echo ]**

**Introduction**

Memory management in Linux involves the efficient allocation, tracking, and release of memory resources. The Linux kernel uses a virtual memory system, where each process is given its own isolated address space, with the kernel mapping it to physical memory. It employs paging to manage memory, dividing it into fixed-size blocks called pages. Linux also uses a combination of swap space and RAM to handle memory overflows. The system ensures that processes get the memory they need while minimizing fragmentation, and it reclaims unused memory through garbage collection techniques like page reclamation.

**Segmented Memory**

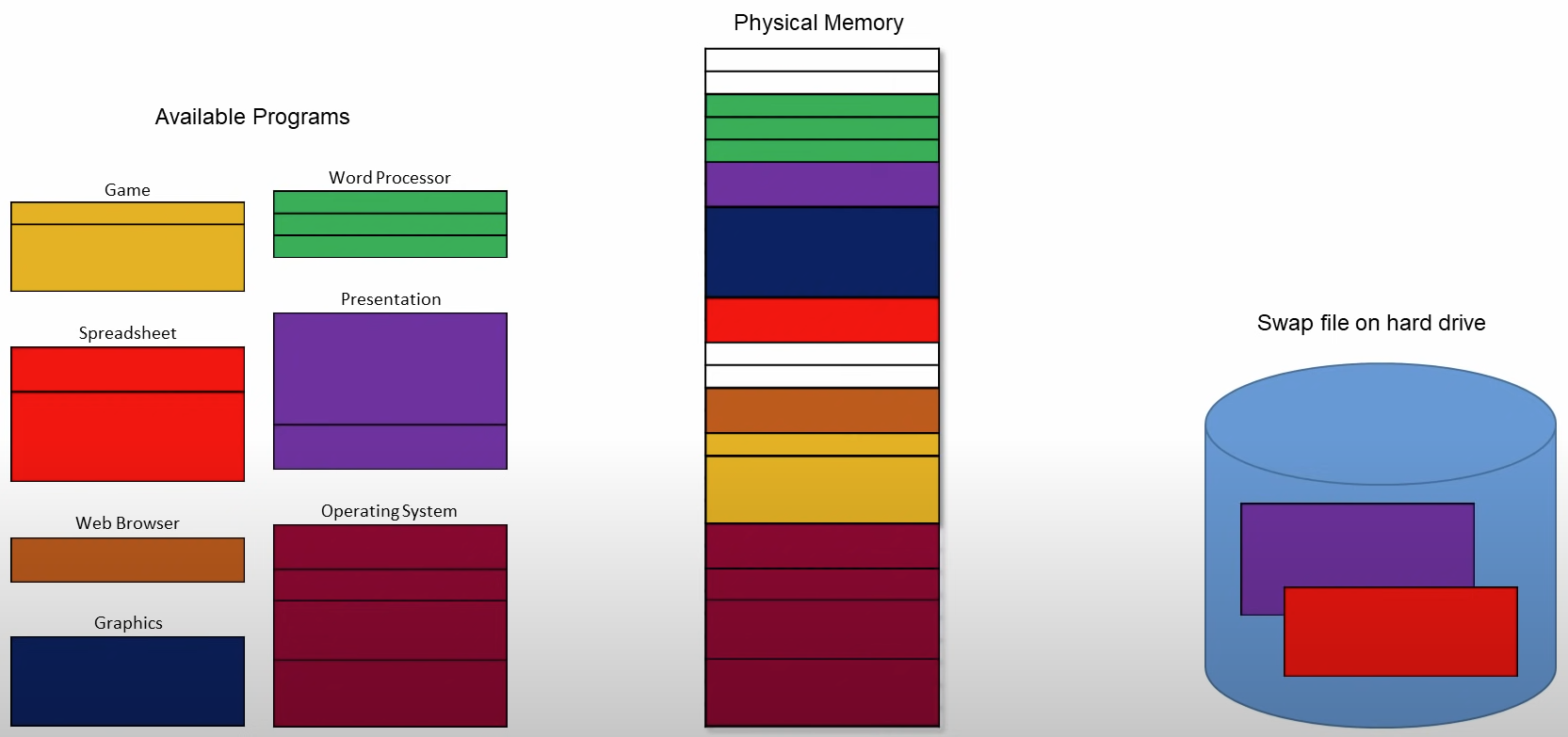
1. Segments are swapped between primary memory (RAM) and secondary memory (ROM)
2. Program segment corresponds to blocks of program code such as procedures or functions.
3. Data segment corresponds to data structures like stack, queue, graph, etc.
4. The OS knows about start and size of the segment in memory.
5. Segmentation can result in memory fragmentation (unused space in between segments).
6. A segment in memory can be only replaced by a segment of the same or smaller size.
7. Large memory segments may not be allowed in the memory very often.

**Paged Memory**

1. Memory is split up into small, equally sized sections called pages (4 Kb in size).
2. A single application may occupy multiple pages that are not necessarily contiguous.
3. Each application has its own view of memory (i.e.) it has its own virtual address space called virtual memory.
4. A page table records where the different pages of a program are located in the physical memory. It is used to map virtual pages to physical memory.
5. The virtual memory is divided into pages and the physical memory is divided into frames.
6. When a process tries to access a page that is not currently in physical memory, a page fault occurs. The operating system then loads the required page from disk into RAM, replacing an existing page if necessary
7. If RAM is full, the operating system swaps out pages that are not actively used to disk (swap space) to free up space for other pages. This allows the system to handle larger workloads than can fit into physical memory at once

**Memory Access in Earlier Days**

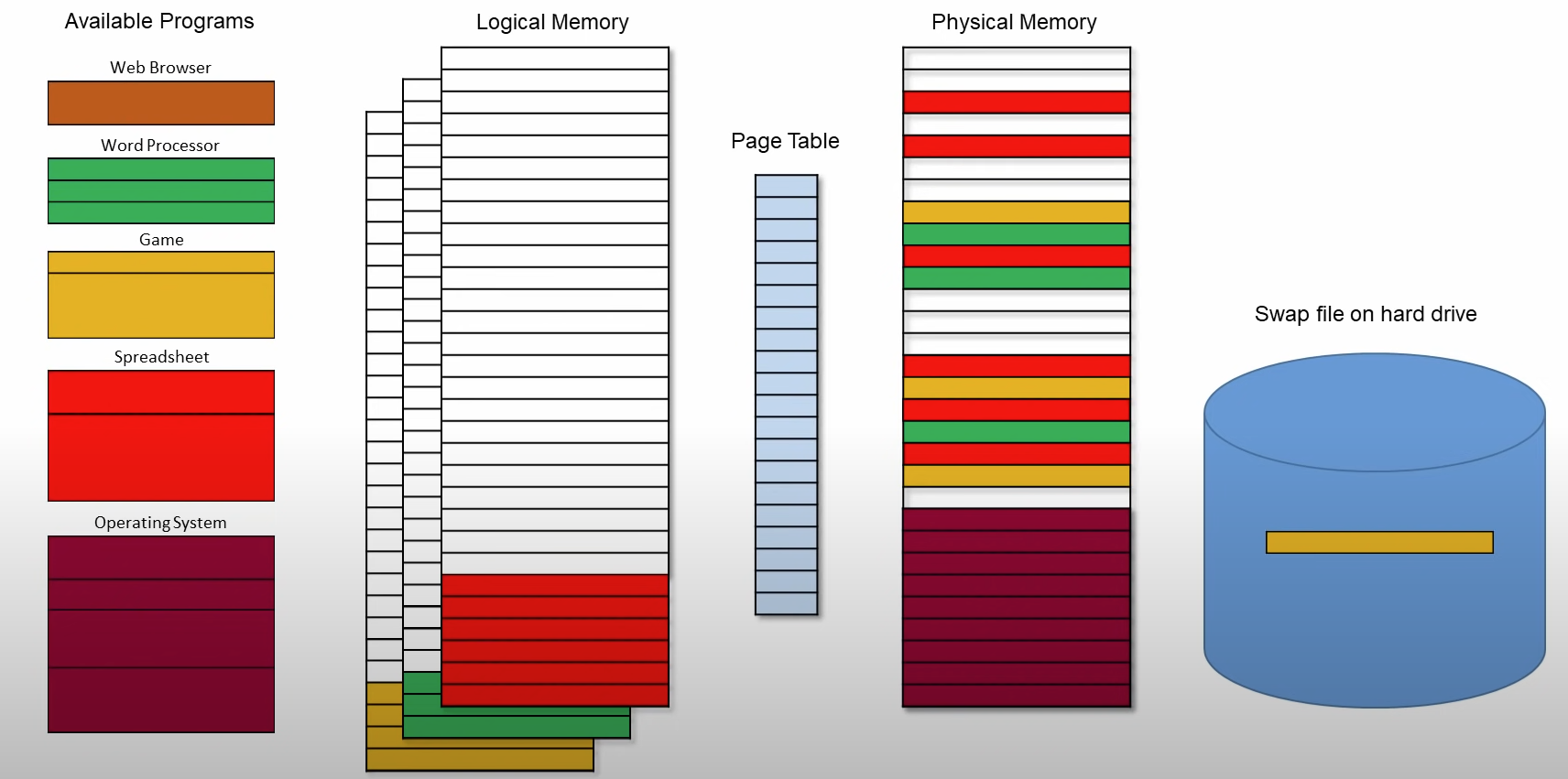
1. When a program (Game, MS Word, MS PPT, etc) is executed, an instance of the running program called a process is created by the kernel.



1. The program data is loaded into the physical memory (RAM) from the HDD/SSD for faster access.
2. The process then occupies the space available in the physical memory (RAM) in contiguous fashion (Segmented Memory) and the OS knows which program requires how much memory and it allocates it to the respective process.
3. The OS also stores the starting segment and ending segment of the process to access the data of the process.
4. If the physical memory is occupied completely, any idle process is sent back to the main memory (HDD/SSD) and the space is given to the new process which asks for the memory, as all processes occupied contiguous space if there was some free space available that was not contiguous it could be allocated to this new process.
5. Also earlier the physical memory had a single address space where all processes occupied its space in the same address space, this made processes corrupt or interfere with the data of other processes.
6. To overcome these issues paged memory was introduced.

**Memory Access at Present Time**

1. When we open google chrome, the OS loads the browser’s essential data (only code required to initiate the program) into physical memory and the rest of the pages are in swap space in the disk.



1. Google chrome has its own virtual address space which is divided into pages.
2. The operating system uses a page table to map virtual addresses to physical memory. The page table keeps track of which virtual pages correspond to which physical frames in RAM.
3. When Chrome needs to access a specific memory location (e.g., loading the user interface, displaying web pages), it generates a virtual memory address. If the required page is already in RAM (cached in physical memory), the system simply accesses the data from the physical memory address.
4. If the page is not in RAM (e.g., it was swapped out to the hard disk or hasn't been loaded yet), the system triggers a page fault. The operating system then loads the missing page into physical memory from the disk (swap space) or another part of RAM, and updates the page table accordingly.

**Components of Memory Management in Linux**

### **1. Virtual Memory**

* The CPU generates virtual addresses for each process.
* The Memory Management Unit (MMU) translates these virtual addresses to physical addresses using page tables.
* Each process gets its own virtual address space, isolating it from other processes.

### **2. Paging**

* Memory is divided into fixed-size pages (typically 4KB).
* When a process accesses data not currently in RAM, a page fault occurs.
* The kernel locates the required page in secondary storage (swap space) and loads it into RAM.
* Least Recently Used (LRU) or similar algorithms are used to decide which pages to be pushed to secondary storage if RAM is full.

### **3. Kernel and User Space**

* At process creation, the kernel assigns memory in user space for application execution.
* The kernel runs in kernel space, handling hardware, I/O, and system calls from user applications.
* Separation ensures stability, as user processes can’t interfere with kernel operations.

### **4. Dynamic Memory Allocation**

* Processes request memory via system calls like malloc( ) or brk( ).
* The kernel allocates memory dynamically using techniques like slab allocation, which provides pre-allocated chunks for objects of similar sizes.
* Allocated memory is tracked and freed when no longer needed.

### **5. Caching and Buffers**

* When a file is accessed, Linux caches it in memory to speed up future reads.
* If free memory is available, the kernel keeps cached data until it’s explicitly needed by an application.
* Data waiting to be written to disk is buffered to improve write performance.

### **7. Memory Zones**

* Physical memory is divided into zones like DMA, Normal, and HighMem.
* During allocation, the kernel checks which zone the request fits and allocates accordingly.
* Hardware-specific allocations (like for DMA devices) come from their designated zones.

### **8. Out-of-Memory (OOM) Killer**

* When free memory and swap space are exhausted, the kernel activates the OOM killer.
* The OOM killer identifies and terminates processes consuming the most memory or those deemed less critical.
* Resources from the terminated process are freed to keep the system operational.

**Free**

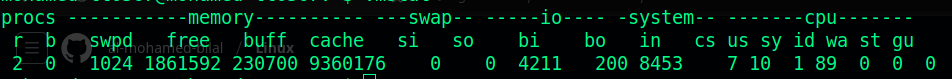
The free command in Linux is used to display the system's memory usage, including physical memory, swap space, and buffers/cache. It provides an overview of how much memory is used, free, shared, and available.

*free <options> (or) free -h*

**Vmstat**

The vmstat (Virtual Memory Statistics) command in Linux is used to report system performance information, focusing on memory, processes, paging, block I/O, traps, and CPU activity. It provides a snapshot of the system’s overall health and can help in troubleshooting performance issues.

*vmstat <options> (or) vmstat -d*

**

**r:** Number of processes waiting for run time (ready to run).

**b:** Number of processes in uninterruptible sleep (waiting for I/O).

**swpd:** Amount of virtual memory used (in KB).

**free:** Amount of idle memory (in KB).

**buff:** Amount of memory used as buffers (in KB).

**cache:** Amount of memory used as cache (in KB).

**si:** Amount of memory swapped in from disk (in KB/s).

**so:** Amount of memory swapped out to disk (in KB/s).

**bi:** Blocks received from a block device (in KB/s).

**bo:** Blocks sent to a block device (in KB/s).

**in:** Number of interrupts per second.

**cs:** Number of context switches per second.

**us:** Percentage of CPU time spent in user space.

**sy:** Percentage of CPU time spent in kernel space.

**id:** Percentage of CPU time spent idle.

**wa:** Percentage of CPU time spent waiting for I/O operations.

**st:** Percentage of CPU time stolen by virtual machine (if applicable).

**Swapon**

The swapon command is used to activate a swap device or file, making it available for use by the system.

*swapon <device> (or) swapon /dev/sda1*

**Swapoff**

The swapoff command is used to deactivate a swap device or file, making it available for use by the system.

*swapoff <device> (or) swapoff /dev/sda1*

**Mkswap**

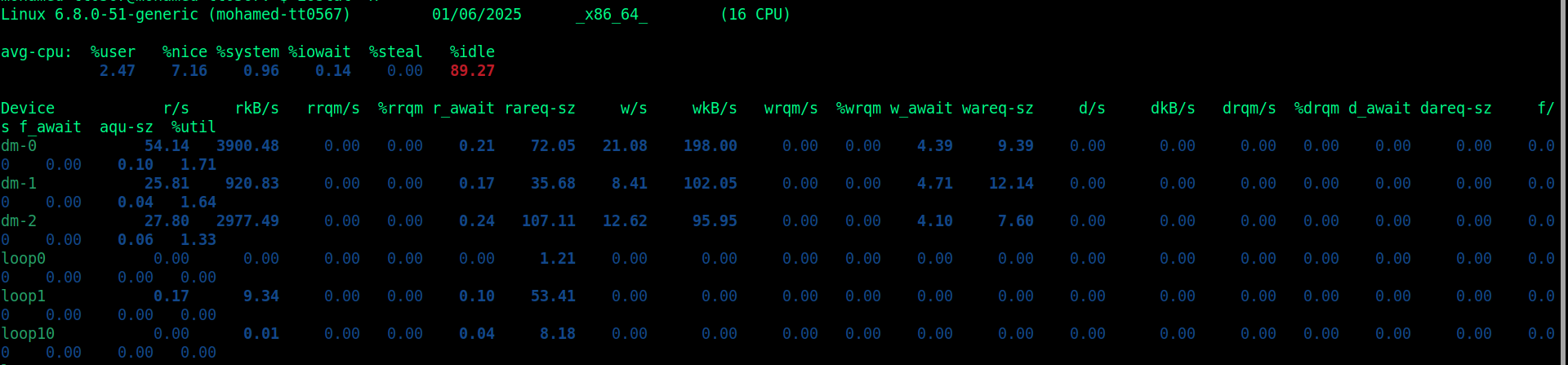
The mkswap command is used to initialize a swap device or file, preparing it to be used as swap space. This command must be run before using the device or file with swapon

*mkswap <device> (or) mkswap /dev/sdc1*

**Iostat**

The iostat (Input/Output Statistics) command in Linux is used to monitor system input/output device loading by reporting the CPU statistics and input/output statistics for devices, partitions, and network filesystems. It provides useful information about system performance and can help in diagnosing bottlenecks, particularly related to disk I/O.

*iostat <options> (or) iostat -x*



**%user:** Percentage of CPU time spent in user space (non-kernel).

**%system:** Percentage of CPU time spent in kernel space (system).

**%idle:** Percentage of CPU time spent in idle mode (not executing any task).

**%iowait:** Percentage of CPU time spent waiting for I/O operations.

**r/s:** Number of read requests per second.

**w/s:** Number of write requests per second.

**rkB/s:** Number of kilobytes read per second.

**wkB/s:** Number of kilobytes written per second.

**avgrq-sz:** Average size (in sectors) of the I/O requests.

**avgqu-sz:** Average queue length of the disk device.

**await:** Average time (in milliseconds) spent by a request in the queue and the device.

**svctm:** Average service time for I/O requests (in milliseconds).

**%util:** Percentage of time the device was busy handling I/O requests.

**Sar**

The sar (System Activity Reporter) command in Linux is used to collect, report, and save system activity information, such as CPU usage, memory usage, disk activity, network statistics, and more. It provides detailed reports on various system resources over time and is useful for performance monitoring, troubleshooting, and system diagnostics.

*sar <options> (or) sar -A (or) sar -d (or) sar -u*

**%user:** CPU time spent in user space.

**%system:** CPU time spent in kernel space.

**%idle:** Time the CPU is idle.

**%iowait:** Time the CPU waits for I/O operations to complete.

**kbmemfree:** Amount of free memory.

**kbmemused:** Amount of used memory.

**kbswpfree:** Free swap space.

**kbswpused:** Used swap space.

**tps:** Transactions per second.

**rd\_sec/s:** Read sectors per second.

**wr\_sec/s:** Write sectors per second.

**await:** Average I/O wait time (ms).

**%util:** Percentage of time the device was busy.

**rxpck/s:** Packets received per second.

**txpck/s:** Packets transmitted per second.

**rxkB/s:** Kilobytes received per second.

**txkB/s:** Kilobytes transmitted per second.

**Sync**

The sync command in Linux is used to flush the file system buffers to disk. This ensures that all data, including data that has been cached in memory, is written to the disk. It helps in making sure that any changes made to files, directories, and other system structures are persisted in the storage device, preventing data loss in case of a system crash or shutdown.

*sync*

**Drop\_Caches**

The drop\_caches command in Linux is used to clear the system's file system caches, freeing up memory that is being used by the kernel to cache data. These caches include:

* **Page Cache:** Cache for frequently accessed file data.
* **Directory Entry Cache:** Cache for directory contents.
* **Inode Cache:** Cache for inodes that describe files on the disk.

1. **Clear PageCache :** Only the file content cache

*echo 1 > /proc/sys/vm/drop\_caches*

1. **Clear dentries and inodes :** Directory entry and inode caches

*echo 2 > /proc/sys/vm/drop\_caches*

1. **Clear PageCache, dentries, and inodes** **:** All caches

*echo 3 > /proc/sys/vm/drop\_caches*