Operating System Concepts

- OS Concepts
- Linux commands
- Shell scripts
- Linux System call Programming

Learning OS

- step 1: End user
  - Linux commands
- step 2: Administrator
  - Install OS (Linux)
  - Configuration - Users, Networking, Storage, ...
  - Shell scripts
- step 3: Programmer
  - Linux System call programming
- step 4: Designer/Internals
  - UNIX & Linux internals

What is OS?

- Interface between end user and computer hardware.
- Interface between Programs and computer hardware.
- Control program that controls execution of all other programs.
- Resource manager/allocator that manage all hardware resources.
- Bootable CD/DVD = Core OS + Applications + Utilities
- Core OS = Kernel -- Performs all basic functions of OS.

OS Functions

- CPU scheduling
- Process Management
- Memory Management
- File & IO Management
- Hardware abstraction
- User interfacing
- Security & Protection
- Networking

Process Management

Program

- Set of instructions given to the computer --> Executable file.
- Program --> Sectioned binary --> "objdump" & "readelf".

  - Exe header --> Magic number, Address of entry-point function, Information about all sections. (objdump -h program.out)
  - Text --> Machine level code (objdump -S program.out)
  - Data --> Global and Static variables (Initialized)
  - BSS --> Global and Static variables (Uninitialized)
  - RoData --> String constants
  - Symbol Table --> Information about the symbols (Name, Size, section, Flags, Address) (objdump -t program.out)
- Program (Executable File) Format
  - Windows -- PE
  - Linux -- ELF
- Program are stored on disk (storage).

Process

- Progam under execution
- Process execute in RAM.
- Process control block contains information about the process (required for the execution of process).
  - Process id
  - Exit status
    - 0 - Indicate successful execution
    - Non-zero - Indicate failure
  - Scheduling information (State, Priority, Sched algorithm, Time, ...)
  - Memory information (Base & Limit, Segment table, or Page table)
  - File information (Open files, Current directory, ...)
  - IPC information (Signals, ...)
  - Execution context (Values of CPU registers)
  - Kernel stack
- PCB is also called as process descriptor (PD), uarea (UNIX), or task_struct (Linux).
- In Linux size of task_struct is approx 4KB

Process

- Process is program in execution.
- Process has multiple sections i.e. text, data, rodata, heap, stack. ... into user space and its metadata is stored into kernel space in form of PCB struct.
- PCB contains
  - id, exit status,
  - scheduling info (state, priority, time left, scheduling policy, ...),
  - files info (current directory, root directory, open file descriptor table, ...),
  - memory information (base & limit, segment table, or page table),
  - ipc information (signals, ...),
  - execution context, kernel stack, ...

File Management

File

- File is collection of data/information on storage device.
  - File = Contents (Data) + Information (Metadata)
  - The data is stored in zero or more Data blocks (in FS), while metadata is stored in the FCB (in filesystem).
- FCB is called as "inode" on UNIX/Linux. It contains
  - type: UNIX/Linux has 7 types of files
    - -: regular, d: directory, l: symbolic link, p: pipe, s: socket, c: char device, b: block device
  - size: number of bytes
  - links: number of hard links
  - mode (permissions): (u) rwx, (g) rwx, (o) rwx
  - user & group
  - time-stamps: modification, creation, access.
  - info about data blocks
- terminal> ls -l
  - type, mode, links, user, group, size, timestamp, name.
- terminal> stat filepath

File System

- Files are stored on storage device. Arrangement of files in storage device is called as "File System".
- e.g. FAT, NTFS, EXT2/3/4, ReiserFS, XFS, HFS, etc.
- File System logically divide partition into 4 sections.
  - Boot block/Boot sector
    - Contains programs/info required for booting of OS
    - Typically contains bootstrap program and bootloader program
  - Super block/Volume control block
    - Contains information of whole partition.
    - Capacity, Label.
    - terminal> df -h
      - Total number of data blocks/inodes.
      - Number of used/free data blocks/inodes.
      - Information of free data blocks/inodes.
  - Inode List/Master file table
    - Inodes (FCB) for each file
  - Data blocks
    - Stores data of the file.
    - Each file have zero or more data blocks.
    - Size of data blocks can be configured while creating file system
- File system is created by the format utility while formatting the partition.
  - Windows: format.exe
  - Linux: mkfs
    - terminal> sudo mkfs -t ext3 /dev/sdb1
    - terminal> sudo mkfs -t vfat /dev/sdb1
    - -t fs_type e.g. ext3, ext4, vfat, ntfs, ...

        - partition e.g. /dev/sdb1

- Disk/partition naming conventions
    - Windows:
        - Disks are named as disk0, disk1, ...
        - partitions are named as drives i.e. C:, D:, E:, ...
    - Linux:
        - Disks are named as /dev/sda, /dev/sdb, /dev/sdc, etc.
        - Partitions per disk are named as
            - sda partitions: sda1, sda2, sda3, ...
            - sdb partitions: sdb1, ...

Linux File Structure

- Linux follows "/" (root) file system.
- "/" is a starting point of Linux file system.
- All your data is stored in this partition.
- / contains boot, bin, sbin, etc, root, home, dev, proc, mnt, media, opt
- In Linux everything is a file.
- Mainly there are two types of files in Linux
    - File
    - Directory (Folder)
- Linux Directories
    - boot - files related to booting
        - vmlinuz - kernel Image
        - grub - boot loader
        - config - kernel configuration
        - initrd/initramfs - initail root file system
    - bin - user commands in binary format
    - sbin - all admin/system commands in binary format
    - etc - configuration files
    - root - home directory of root user
    - home - it contains sub directories for each user with its name
        - devendra -> /home/devendra
        - sunbeam -> /home/sunbeam
        - osboxes -> /home/osboxes
    - dev - it contains all device related files
    - lib - shared program libraries required by kernel
    - mnt - it is temporary mount point
    - media - it is mount point for media eg cdrom
    - opt - stores optional files of large softwares
    - proc - virtual file system - it contains information about system or processes
    - sys - entries of each block devices, subdirectories for each physical bus type supported, every device class registered with the kernel, global device hierarchy of all devices
    - tmp - temporary files that may be lost on system shutdown
    - usr - read only directory that stores small programs and files accessible to all users

User interfacing

- UI of OS is a program (Shell) that interface between End user and Kernel.
- Shell -- Commmand interpreter
    - End user --> Command --> Shell --> Kernel
- User interfacing (Shell)
    - Graphical User Interface (GUI)
    - Command Line Interface (CLI)

Example shells

- Windows
    - GUI shell: explorer.exe
    - CLI shell: cmd.exe, powershell.exe
- DOS
    - CLI shell: command.com
- Unix/Linux
    - CLI shell: bsh, "bash", ksh, csh, zsh, ...
        - ls /bin/*sh
        - echo $SHELL
    - shell of current user can be changed using "chsh" command.
- GUI shell/standards
    - GNOME: GNU Network Object Model Environment (e.g. Ubuntu, Redhat, CentOS, ...)
    - KDE: Kommon Desktop Environment (e.g. Kubuntu, SuSE, ...)

Path

- It is a unique location of any file in the file system.
- It is represented by character strings with few delimiters (“/”, “\”, “:”)
- Types of path
    - There are two types of paths in linux
    - Absolute path
        - Path which starts with “/” is called as absolute path.
        - E.g. /home/devendra/MyData/Demos/demo01.sh
    - Relative path
        - Path with respect to current directory is called as relative path
        - E.g. MyData/Assignments/assign02.pdf

Linux Admin

- In Linux, Admin is called as "super-user".
- Admin's login name is "root".
- Most of modern Linux, disable "root" login (for security).
- To execute commands with admin privileges use "sudo" (if approved by system admin).
    - cmd> sudo apt update
    - cmd> sudo apt install vim gcc python3 python3-pip
    - cmd> sudo snap install --classic code

Directory commands

- pwd -- print present working directory (current directory)
- cd -- change directory (syntax> cd dirpath)
- ls - list directory contents (syntax> ls dirpath)
- mkdir -- make directory (syntax> mkdir dirpath)
- rmdir -- remove empty directory (syntax> rmdir dirpath)
- cd
    - cd ~ - change working directory to home directory
    - cd - - change working directory to old working directory
    - cd .. - change working directory to parent directory

File commands

- cat
    - cat > filepath <-- create new file
    - cat filepath <-- display file contents
- rm
    - rm filepath <-- delete given file
    - rm -r dirpath <-- delete dir with all contents
- mv
    - mv filepath destdirpath <-- move given file into given dest directory
    - mv dirpath destdirpath <-- move given dir into given dest directory
    - mv oldname newname <-- rename given file
- cp
    - cp filename newfilename <-- copy file with new name/path.
    - cp filepath destdirpath <-- copy file into given dest dir with same name.
    - cp -r dirpath destdirpath <-- copy file into given dest dir with same name.

Linux commands

- cd
    - cd ~ - change working directory to home directory
    - cd - - change working directory to old working directory
    - cd .. - change working directory to parent directory
- ls

  - ls - list the contents of present working directory
  - ls path - list the contents of given path
  - ls -l - list the contents in detail format
    - type and permissions
      - Types of files
        - Regular file (-)
        - Directory file (d)
        - Link file (l)
        - pipe file (p)
        - socket file (s)
        - character special file (c)
        - block specical file (b)
      - Permissions of files
        - r - read, w - write, x - execute
        - (rwx)user/owner, (rwx)group, (rwx)others
    - link count
    - user/owner
    - group
    - size
    - timestamp
    - name
  - ls -a - list all contents along with hidden
  - ls -A - list all contents along with hidden except . and ..
  - ls -i - list contents with indoe number
    - inode number is unique number given to every file
  - ls -s - list content with size (number of blocks)
  - ls -S - list content in descending order of their sizes

- touch

  - if file does not exist, empty file is created
  - if file exist, timestamp of that file is changed

- stat

  - stat file - display information of file
  - stat file1 file2 - display information of file1 and file2
  - stat -c "format" file - display file information in given format

- head

  - head file - display first 10 lines
  - head -5 file - display first 5 lines

- tail

  - tail file - display last 10 lines
  - tail -4 file - display last 4 lines

- sort

    - sort file - sort the content by alphabetically
    - sort -n file - sort the content by their value
        - sort command do not modify file content

- uniq
    - uniq file - display contents uniquely (truncate duplicate)
        - truncate duplicate content if it is consecutive

- rev filepath
    - Print each line reversed.
    - File contents are not modified.

- tac filepath
    - Print all lines in reverse order. The first line printed at last, while last line printed first.

- stat path
    - Display info about file or directory.

- alias
    - alias list="ls -l"
        - list will be alias/nick name to ls -l
        - list will give output same as ls -l

- unalias
    - unalias list

- which
    - which command
        - display the location of command executable.

- whereis
    - whereis command
        - display the location of command executable and also manual paage location.

Redirection

- for every command input is taken from terminal, output is printed on terminal and error is also printed on terminal
- Standard streams (by defualt for every process, three files are opened)
    - stdin
    - stdout
    - stderr
- There are three types of redirections
    - input redirection

        - input will be taken from file instead of stdin
        - to do input redirection '<' symbol is used
        - command < file

    - output redirection

        - output will be written into file instead of stdout
        - to do output redirection '>' or '>>' symbol is used
        - command > file
            - older content of file will be over written
        - command >> file
            - content will be appneded into file at the end

    - error redirection

        - error will be written into file instead of stderr
        - to do output redirection '2>' or '2>>' symbol is used
        - command 2> file
            - older content of file will be over written
        - command 2>> file
            - content will be appneded into file at the end

Pipe

- Using pipe, we can redirect output of any command to the input of any other command.
- Two processes are connected using pipe operator (|).
- Two processes runs simultaneously and are automatically rescheduled as data flows between them.
- If you don't use pipes, you must use several steps to do single task.
- command1 | command2
    - output of command1 will be given as input to command 2
- E.g.
    - who | wc

Shell meta characters

- '*' - zero or more occurances of any character
- '?' - one occurance of any character

Regular Expressions

- Find a pattern in text file(s).
- Regular expressions are patterns used to match character combinations in strings.
- A regular expression pattern is composed of simple characters, or a combination of simple and special characters e.g. /abc/, /ab*c/
- Pattern is given using regex wild-card characters.
    - Basic wild-card characters
        - $ - find at the end of line.
        - ^ - find at the start of line.
        - [ ] - any single char in give range or set of chars

    - [^] - any single char not in give range or set of chars
    - . - any single character
    -
      -
        - zero or more occurrences of previous character
  - Extended wild-card characters
    - ? - zero or one occurrence of previous character
    -
      -
        - one or more occurrences of previous character
    - {n} - n occurrences of previous character
    - {,n} - max n occurrences of previous character
    - {m,} - min m occurrences of previous character
    - {m,n} - min m and max n occurrences of previous character
    - () - grouping (chars)
    - (|) - find one of the group of characters

grep

- Regex commands
  - grep - GNU Regular Expression Parser - Basic wild-card
  - egrep - Extended Grep - Basic + Extended wild-card
  - fgrep - Fixed Grep - No wild-card
- Command syntax
  - grep "pattern" filepath
  - grep [options] "pattern" filepath
    - -c : count number of occurrences
    - -v : invert the find output
    - -i : case insensitive search
    - -w : search whole words only
    - -R : search recursively in a directory
    - -n : show line number.

Classification of OS

- OS can be categorized based on the target system (computers).
  - Desktop systems
  - Server systems
  - Embedded systems
  - Distributed systems
  - Hand-held systems
  - Real-time systems

Desktop systems

- Personal computers -- desktop and laptops
- User convinence and Responsiveness
- Examples: Windows, Mac, Linux, few UNIX, ...

Handheld systems

- OS installed on handheld devices like mobiles, PDAs, iPODs, etc.

- Challenges:
  - Small screen size
  - Low end processors
  - Less RAM size
  - Battery powered
- Examples: Symbian, iOS, Linux, PalmOS, WindowsCE, etc.

Realtime systems

- The OS in which accuracy of results depends on accuracy of the computation as well as time duration in which results are produced, is called as "RTOS".
- If results are not produced within certain time (deadline), catastrophic effects may occur.
- These OS ensure that tasks will be completed in a definite time duration.
- Time from the arrival of interrupt till begin handling of the interrupt is called as "Interrupt Latency".
- RTOS have very small and fixed interrupt latencies.
- RTOS Examples: uC-OS, VxWorks, pSOS, RTLinux, FreeRTOS, etc.

Distributed systems

- Multiple computers connected together in a close network is called as "distributed system".
- Its advantages are high availability (24x7), high scalability (many clients, huge data), fault tolerance (any computer may fail).
- The requests are redirected to the computer having less load using "load balancing" techniques.
- The set of computers connected together for a certain task is called as "cluster". Examples: Linux.

Classification of OS

Resident Monitor

- Early (oldest) OS resides in memory and monitor execution of the programs. If it fails, error is reported.
- OS provides hardware interfacing that can be reused by all the programs.

Batch Systems

- The batch/group of similar programs is loaded in the computer, from which OS loads one program in the memory and execute it. The programs are executed one after another.
- In this case, if any process is performing IO, CPU will wait for that process and hence not utilized efficiently.

Multi-Programming

- In multi-programming systems, multiple program can be loaded in the memory.
- The number of program that can be loaded in the memory at the same time, is called as "degree of multi-programming".
- In these systems, if one of the process is performing IO, CPU can continue execution of another program. This will increase CPU utilization.
- Each process will spend some time for CPU computation (CPU burst) and some time for IO (IO burst).
  - If CPU burst > IO burst, then process is called as "CPU bound".
  - If IO burst > CPU burst, then process is called as "IO bound".
- To efficiently utilize CPU, a good mix of CPU bound and IO bound processes should be loaded into memory. This task is performed by an unit of OS called as "Job scheduler" OR "Long term scheduler".
- If multiple programs are loaded into the RAM by job scheduler, then one of process need to be executed (dispatched) on the CPU. This selection is done by another unit of OS called as "CPU scheduler" OR "Short term scheduler".

Multi-tasking OR time-sharing

- CPU time is shared amoung multiple processes in the main memory is called as "multi-tasking".
- In such system, a small amount of CPU time is given to each process repeatedly, so that response time for any process < 1 sec.
- With this mechanism, multiple tasks (ready for execution) can execute concurrently.
- There are two types of multi-tasking:
  - Process based multitasking: Multiple independent processes are executing concurrently. Processes running on multiple processors called as "multi-processing".
  - Thread based multi-tasking OR multi-threading: Multiple parts/functions in a process are executing concurrently.

Multiprocessor systems

- The systems in which multiple processors are connected in a close circuit is called as "multiprocessor computer".
- The programs/OS take advantage of multiple processors in the computer are called as "Multiprocssing" programs/OS.

- Windows Vista: First Windows OS designed for multi-processing.
- Linux 2.5+: Linux started supporting multi-processing.

- Modern PC architectures are multi-core arch i.e. multiple CPUs on single chip.
- Since multiple tasks can be exeuted on these processors simultaneously, such systems are also called as "parallel systems".
- Parallel systems have more throughput (Number of tasks done in unit time).
- There are two types of multiprocessor systems:
  - Asymmetric Multi-processing
  - Symmetric Multi-processing

Asymmetric Multi-processing

- OS treats one of the processor as master processor and schedule task for it. The task is in turn divided into smaller tasks and get them done from other processors.

Symmetric Multi-processing

- OS considers all processors at same level and schedule tasks on each processor individually.
- All modern desktop systems are SMP.

Multi-user

- Multiple users can execute multiple tasks concurrently on the same systems. e.g. IBM 360, UNIX, Windows Servers, etc.
- Each user can access system via different terminal.
- There are many UNIX commands to track users and terminals.
  - tty, who, who am i, whoami, w

Process Life Cycle

Process States

- New
    - New process PCB is created and added into job queue. PCB is initialized and process get ready for execution.
- Ready
    - The ready process is added into the ready queue. Scheduler pick a process for scheduling from ready queue and dispatch it on CPU.
- Running
    - The process runs on CPU. If process keeps running on CPU, the timer interrupt is used to forcibly put it into ready state and allocate CPU time to other process.
- Waiting
    - If running process request for IO device, the process waits for completion of the IO. The waiting state is also called as sleeping or blocked state.
- Terminated
    - If running process exits, it is terminated.
- Linux: TASK_RUNNING (R), TASK_INTERRUPTIBLE (S), TASK_UNINTERRUPTIBLE (D), TASK_STOPPED(T), TASK_ZOMBIE (Z), TASK_DEAD (X)

Types of Scheduling

Non-preemptive

- The current process gives up CPU voluntarily (for IO, terminate or yield).
- Then CPU scheduler picks next process for the execution.
- If each process yields CPU so that other process can get CPU for the execution, it is referred as "Co-operative scheduling".

Preemptive

- The current process may give up CPU voluntarily or paused forcibly (for high priority process or upon completion of its time quantum)

Scheduling criteria's

CPU utilization: Ideal - max

- On server systems, CPU utilization should be more than 90%.
- On desktop systems, CPU utilization should around 70%.

Throughput: Ideal - max

- The amount of work done in unit time.

Waiting time: Ideal - min

- Time spent by the process in the ready queue to get scheduled on the CPU.
- If waiting time is more (not getting CPU time for execution) -- Starvation.

Turn-around time: Ideal - CPU burst + IO burst

- Time from arrival of the process till completion of the process.
- CPU burst + IO burst + (CPU) Waiting time + IO Waiting time

Response time: Ideal - min

- Time from arrival of process (in ready queue) till allocated CPU for first time.

Scheduling Algorithms

FCFS

- Process added first in ready queue should be scheduled first.
- Non-preemptive scheduling
- Scheduler is invoked when process is terminated, blocked or gives up CPU is ready for execution.
- Convoy Effect: Larger processes slow down execution of other processes.

SJF

- Process with lowest burst time is scheduled first.
- Non-preemptive scheduling
- Minimum waiting time

SRTF - Shortest Remaining Time First

- Similar to SJF - but Preemptive scheduling
- Minimum waiting time

Priority

- Each process is associated with some priority level. Usually lower the number, higher is the priority.
- Preemptive scheduling or Non Preemptive scheduling
- Starvation
  - Problem may arise in priority scheduling.
  - Process not getting CPU time due to other high priority processes.
  - Process is in ready state (ready queue).
  - May be handled with aging -- dynamically increasing priority of the process.

Round-Robin

- Preemptive scheduling

- Process is assigned a time quantum/slice.
- Once time slice is completed/expired, then process is forcibly preempted and other process is scheduled.
- Min response time.

Thread concept

- Threads are used to execute multiple tasks concurrently in the same program/process.
- Thread is a light-weight process.
    - For each thread new control block and stack is created. Other sections (text, data, heap, ...) are shared with the parent process.
    - Inter-thread communication is much faster than inter-process communication.
    - Context switch between two threads in the same process is faster.
- Thread stack is used to create function activation records of the functions called/executed by the thread.

Process vs Thread

- In modern OS, process is a container holding resources required for execution, while thread is unit of execution/scheduling.
- Process holds resources like memory, open files, IPC (e.g. signal table, shared memory, pipe, etc.).
- PCB contains resources information like pid, exit status, open files, signals/ipc, memory info, etc.
- CPU time is allocated to the threads. Thread is unit of execution.
- TCB contains execution information like tid, scheduling info (priority, sched algo, time left, ...), Execution context, Kernel stack, etc.
- terminal> ps -e -o pid,nlwp,cmd
- terminal> ps -e -m -o pid,tid,nlwp

main thread

- For each process one thread is created by default called as main thread.
- The main thread executes entry-point function of the process.
- The main thread use the process stack.
- When main thread is terminated, the process is terminated.
- When a process is terminated, all threads in the process are terminated.

Memory Management

- In multi-programming OS, multiple programs are loaded in memory.
- RAM memory should be divided for multiple processes running concurrently.
- Memory Mgmt scheme used by any OS depends on the MMU hardware used in the machine.
- There are three memory management schemes are available (as per MMU hardware).
    1. Contiguous Allocation
    2. Segmentation
    3. Paging

Contiguous Allocation

Fixed Partition

- RAM is divided into fixed sized partitions.
- This method is easy to implement.
- Number of processes are limited to number of partitions.
- Size of process is limited to size of partition.
- If process is not utilizing entire partition allocated to it, the remaining memory is wasted. This is called as "internal fragmentation".

Dynamic Partition

- Memory is allocated to each process as per its availability in the RAM. After allocation and deallocation of few processes, RAM will have few used slots and few free slots.
- OS keep track of free slots in form of a table.
- For any new process, OS use one of the following mechanism to allocate the free slot.
    - First Fit: Allocate first free slot which can accommodate the process.
    - Best Fit: Allocate that free slot to the process in which minimum free space will remain.
    - Worst Fit: Allocate that free slot to the process in which maximum free space will remain.
- Statistically it is proven that First fit is faster algo; while best fit provides better memory utilization.
- Memory info (physical base address and size) of each process is stored in its PCB and will be loaded into MMU registers (base & limit) during context switch.
- CPU request virtual address (address of the process) and is converted into physical address by MMU as shown in diag.
- If invalid virtual address is requested by the CPU, process will be terminated.
- If amount of memory required for a process is available but not contiguous, then it is called as "external fragmentation".
- To resolve this problem, processes in memory can be shifted/moved so that max contiguous free space will be available. This is called as "compaction".

Virtual Memory

- The portion of the hard disk which is used by OS as an extension of RAM, is called as "virtual memory".
- If sufficient RAM is not available to execute a new program or grow existing process, then some of the inactive process is shifted from main memory (RAM), so that new program can execute in RAM (or existing process can grow). It is also called as "swap area" or "swap space".

- Shifting a process from RAM to swap area is called as "swap out" and shifting a process from swap to RAM is called as "swap in".
- In few OS, swap area is created in form of a partition. E.g. UNIX, Linux, ...
- In few OS, swap area is created in form of a file E.g. Windows (pagefile.sys), ...
- Virtual memory advantages:
    - Can execute more number of programs.
    - Can execute bigger sized programs.

Segmentation

- Instead of allocating contiguous memory for the whole process, contiguous memory for each segment can be allocated. This scheme is known as "segmentation".
- Since process does not need contiguous memory for entire process, external fragmentation will be reduced.
- In this scheme, PCB is associated with a segment table which contains base and limit (size) of each segment of the process.
- During context switch these values will be loaded into MMU segment table.
- CPU request virtual address in form of segment address and offset address.
- Based on segment address appropriate base-limit pair from MMU is used to calculate physical address as shown in diag.
- MMU also contains STBR register which contains address of current process's segment table in the RAM.

Demand Segmentation

- If virtual memory concept is used along with segmentation scheme, in case low memory, OS may swap out a segment of inactive process.
- When that process again start executing and ask for same segment (swapped out), the segment will be loaded back in the RAM. This is called as "demand segmentation".
- Each entry of the segment table contains base & limit of a segment. It also contains additional bits like segment permissions, valid bit, dirty bit, etc
- If segment is present in main memory, its entry in seg table is said to be valid (v=1). If segment is swapped out, its entry in segment table is said to be invalid (v=0).

Paging

- RAM is divided into small equal sized partitions called as "frames" / "physical pages".
- Process is divided into small equal sized parts called as "pages" or "logical/virtual pages".
- page size = frame size.
- One page is allocated to one empty frame.
- OS keep track of free frames in form of a linked list.
- Each PCB is associated with a table storing mapping of page address to frame address. This table is called as "page table".
- During context switch this table is loaded into MMU.
- CPU requests a virtual address in form of page address and offset address. It will be converted into physical address as shown in diag.
- MMU also contains a PTBR, which keeps address of page table in RAM.

- If a page is not utilizing entire frame allocated to it (i.e. page contents are less than frame size), then it is called as "internal fragmentation".
- Frame size can be configured in the hardware. It can be 1KB, 2KB or 4KB, ...
- Typical Linux and Windows OS use page size = 4KB.

Page table entry

- Each PTE is of 32-bit (on x86 arch) and it contains
    - Frame address
    - Permissions (read or write)
    - Validity bit
    - Dirty bit
    - ...

TLB (Translation Look-Aside Buffer) Cache

- TLB is high-speed associative cache memory used for address translation in paging MMU.
- TLB has limited entries (e.g. in P6 arch TLB has 32 entries) storing recently translated page address and frame address mappings.
- The page address given by CPU, will be compared at once with all the entries in TLB and corresponding frame address is found.
- If frame address is found (TLB hit), then it is used to calculate actual physical address in RAM (as shown in diag).
- If frame address is not found (TLB miss), then PTBR is used to access actual page table of the process in the RAM (associated with PCB). Then page-frame address mapping is copied into TLB and thus physical address is calculated.
- If CPU requests for the same page again, its address will be found in the TLB and translation will be faster

Two Level Paging

- Primary page table has number of entries and each entry point to the secondary page table page.
- Secondary page table has number of entries and each entry point to the frame allocated for the process.
- Virtual address requested by a process is 32 bits including
    - p1 (10 bits) -> Primary page table index/addr
    - p2 (10 bits) -> Secondary page table index/addr
    - d (12 bits) -> Frame offset

Demand Paging

- When virtual memory is used with paging memory management, pages can be swapped out in case of low memory.
- The pages will be loaded into main memory, when they are requested by the CPU. This is called as "demand paging".
    - Swapped out pages
    - Pages from program image (executable file on disk)
    - Dynamically allocated pages

Virtual pages vs Logical pages

- By default all pages of user space process can be swapped out/in. This may change physical address of the page. All such pages whose physical address may change are referred as "Virtual pages".
- Few kernel pages are never swapped out. So their physical address remains same forever. All such pages whose physical address will not change are referred as "Logical pages".

Thrashing

- If number of programs are running in comparatively smaller RAM, a lot of system time will be spent into page swapping (paging) activity.
- Due to this overall system performance is reduced.
- The problem can be solved by increasing RAM size in the machine.

Page Fault

- Each page table entry contains frame address, permissions, dirty bit, valid bit, etc.
- If page is present in main memory its page table entry is valid (valid bit = 1).
- If page is not present in main memory, its page table entry is not valid (valid bit = 0).
- This is possible due to one of the following reasons:
  - Page address is not valid (dangling pointer).
  - Page is on disk/swapped out.
  - Page is not yet allocated.
- If CPU requests a page that is not present in main memory (i.e. page table entry valid bit=0), then "page fault" occurs.
- Then OS's page fault exception handler is invoked, which handles page faults as follows:
  1. Check virtual address due to which page fault occured. If it is not valid (i.e. dangling pointer), terminate the process (sending SEGV signal). (Validity fault).
  2. Check if read-write operation is permitted on the address. If not, terminate the process (sending SEGV signal). (Protection fault).
  3. If virtual address is valid (i.e. page is swapped out), then locate one empty frame in the RAM.
  4. If page is on swap device or hard disk, swap in the page in that frame.
  5. Update page table entry i.e. add new frame address and valid bit = 1 into PTE.
  6. Restart the instruction for which page fault occurred.

Page Replacement Algorithms

- While handling page fault if no empty frame found (step 3), then some page of any process need to be swapped out. This page is called as "victim" page.
- The algorithm used to decide the victim page is called as "page replacement algorithm".
- There are three important page replacement algorithms.
  - FIFO
  - Optimal
  - LRU

FIFO

- The page brought in memory first, will be swapped out first.

- Sometimes in this algorithm, if number of frames are increased, number of page faults also increase.
- This abnormal behaviour is called as "Belady's Anomaly".

OPTIMAL

- The page not required in near future is swapped out.
- This algorithm gives minimum number of page faults.
- This algorithm is not practically implementable.

LRU

- The page which not used for longer duration will be swapped out.
- This algorithm is used in most OS like Linux, Windows, ...
- LRU mechanism is implemented using "stack based approach" or "counter based approach".
- This makes algorithm implementation slower.
- Approximate LRU algorithm close to LRU, however is much faster.

Dirty Bit

- Each entry in page table has a dirty bit.
- When page is swapped in, dirty bit is set to 0.
- When write operation is performed on any page, its dirty bit is set to 1. It indicate that copy of the page in RAM differ from the copy in swap area.
- When such page need to be swapped out again, OS check its dirty bit. If bit=0 (page is not modified) actual disk IO is skipped and improves performance of paging operation.
- If bit=1 (page is modified), page is physically overwritten on its older copy in the swap area.

Process Creation

- System Calls
  - Windows: CreateProcess()
  - UNIX: fork()
  - BSD UNIX: fork(), vfork()
  - Linux: clone(), fork(), vfork()

fork() syscall

- To execute certain task concurrently we can create a new process (using fork() on UNIX).
- fork() creates a new process by duplicating calling process.
- The new process is called as "child process", while calling process is called as "parent process".
- "child" process is exact duplicate of the "parent" process except few points pid, parent pid, etc.
- pid = fork();
  - On success, fork() returns pid of the child to the parent process and 0 to the child process.
  - On failure, fork() returns -1 to the parent.
- Even if child is copy of the parent process, after its creation it is independent of parent and both these processes will be scheduled sepeately by the scheduler.
- Based on CPU time given for each process, both processes will execute concurrently.

How fork() return two values i.e. in parent and in child?

- fork() creates new process by duplicating calling process.
- The child process PCB & kernel stack is also copied from parent process. So child process has copy of execution context of the parent.
- Now fork() write 0 in execution context (r0 register) of child process and child's pid into execution context (r0 register) of parent process.
- When each process is scheduled, the execution context will be restored (by dispatcher) and r0 is return value of the function.

getpid() vs getppid()

- pid1 = getpid(); // returns pid of the current process
- pid2 = getppid(); // returns pid of the parent of the current process

When fork() will fail?

- When no new PCB can be allocated, then fork() will fail.
- Linux has max process limit for the system and the user. When try to create more processes, fork() fails.
- terminal> cat /proc/sys/kernel/pid_max

Orphan process

- If parent of any process is terminated, that child process is known as orphan process.
- The ownership of such orphan process will be taken by "init" process.

Zombie process

- If process is terminated before its parent process and parent process is not reading its exit status, then even if process's memory/resources is released, its PCB will be maintained. This state is known as "zombie state".
- To avoid zombie state parent process should read exit status of the child process. It can be done using wait() syscall.

wait() syscall

- ret = wait(&s);
  - arg1: out param to get exit code of the process.
  - returns: pid of the child process whose exit code is collected.
- wait() performs 3 steps:
  - Pause execution parent until child process is terminated.
  - Read exit code from PCB of child process & return to parent process (via out param).
  - Release PCB of the child process.
- The exit status returned by the wait() contains exit status, reason of termination and other details.
- Few macros are provided to access details from the exit code.
  - WEXITSTATUS()

waitpid() syscall

- This extended version of wait() in Linux.
- ret = waitpid(child_pid, &s, flags);
    - arg1: pid of the child for which parent should wait.
        - -1 means any child.
    - arg2: out param to get exit code of the process.
    - arg3: extra flags to define behaviour of waitpid().
- returns: pid of the child process whose exit code is collected.
    - -1: if error occurred.

exec() syscall

- exec() syscall "loads a new program" in the calling process's memory (address space) and replaces the older (calling) one.
- If exec() succeed, it does not return (rather new program is executed).
- There are multiple functions in the family of exec():
    - execl(), execlp(), execle(),
    - execv(), execvp(), execve(), execvpe()
- exec() family multiple functions have different syntaxes but same functionality.

Synchronization

- Multiple processes accessing same resource at the same time, is known as "race condition".
- When race condition occurs, resource may get corrupted (unexpected results).
- Peterson's problem, if two processes are trying to modify same variable at the same time, it can produce unexpected results.
- Code block to be executed by only one process at a time is referred as Critical section. If multiple processes execute the same code concurrently it may produce undesired results.
- To resolve race condition problem, one process can access resource at a time. This can be done using sync objects/primitives given by OS.
- OS Synchronization objects are:
    - Semaphore, Mutex

Semaphore

- Semaphore is a sync primitive given by OS.
- Internally semaphore is a counter. On semaphore two operations are supported:
    - wait operation: dec op: P operation:
        - semaphore count is decremented by 1.
        - if cnt < 0, then calling process is blocked.
        - typically wait operation is performed before accessing the resource.
    - signal operation: inc op: V operation:
        - semaphore count is incremented by 1.
        - if one or more processes are blocked on the semaphore, then one of the process will be resumed.
        - typically signal operation is performed after releasing the resource.
- Semaphore types
    - Counting Semaphore
        - Allow "n" number of processes to access resource at a time.

        - Or allow "n" resources to be allocated to the process.
    - Binary Semaphore
        - Allows only 1 process to access resource at a time or used as a flag/condition.

Mutex

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is same as "binary semaphore".
- Mutex can be unlocked by the same process/thread, which had locked it.

Semaphore vs Mutex

- S: Semaphore can be decremented by one process and incremented by same or another process.
- M: The process locking the mutex is owner of it. Only owner can unlock that mutex.
- S: Semaphore can be counting or binary.
- M: Mutex is like binary semaphore. Only two states: locked and unlocked.
- S: Semaphore can be used for counting, mututal exclusion or as a flag.
- M: Mutex can be used only for mutual exclusion.

Deadlock

- Deadlock occurs when four conditions/characteristics hold true at the same time.
    - No preemption: A resource should not be released until task is completed.
    - Mutual exclusion: Resources is not sharable.
    - Hold & Wait: Process holds a resource and wait for another resource.
    - Circular wait: Process P1 holds a resource needed for P2, P2 holds a resource needed for P3 and P3 holds a resource needed for P1.

Deadlock Prevention

- OS syscalls are designed so that at least one deadlock condition does not hold true.
- In UNIX multiple semaphore operations can be done at the same time.

Deadlock Avoidance

- Processes declare the required resources in advanced, based on which OS decides whether resource should be given to the process or not.
- Algorithms used for this are:
    - Resource allocation graph: OS maintains graph of resources and processes. A cycle in graph indicate circular wait will occur. In this case OS can deny a resource to a process.
    - Banker's algorithm: A bank always manage its cash so that they can satisfy all customers.
    - Safe state algorithm: OS maintains statistics of number of resources and number processes. Based on stats it decides whether giving resource to a process is safe or not (using a formula):
        - Max num of resources required < Num of resources + Num of processes
            - If condition is true, deadlock will never occur.
            - If condition is false, deadlock may occur