Operating System Concepts - Notes

- **Operating System Concepts & Linux Programming:**
  - **Operating System Concepts**
    - Introduction to OS
    - Introduction to Comp Hardware
    - Process Mgmt
    - CPU scheduling
    - Memory Mgmt
    - File & IO Mgmt
  - **Linux Programming**
    - Linux commands
    - Shell Scripts
    - System Calls
  - **What is Operating System?**

- OS is intermediary between application programs and computer hardware.
- OS is a resource allocator, which allocates resources (like CPU, RAM, disk, etc) to the running programs as per their requirements.
- OS is a control program, which controls execution of other programs.
- OS CD/DVD = Core OS + Utilities & Applications
- The part of OS which performs basic minimal functionality is called as "Core OS" or "KERNEL".

- **OS Functionalities:**
  - **Process management:**

- Program is a set of instruction given to the machine.
- It is an executable file.
- This file contains exe header, text section, data section, symbol table, etc.
- The exe header contains:
  - magic number : to identify OS in which program can be executed.
  - address of entry point function.
  - info about all the sections in the file.
- When program is executed, the loader (unit of OS) will load it from the disk into the RAM.
- The loader also verifies magic number and file format and if found invalid, program will be terminated.
- For execution of the program OS allocate some memory in RAM which is logically divided into Text, Data, Stack and Heap sections.
- This is called as a "Process". It is formally defined as "Program in Execution".
- To keep info about the process, OS create one more structure, called as "Process Control Block". This struct contains lot of details like:
  - Process Id
  - Exit status
  - Scheduling Info : state, priority, ...
  - Memory Info : page table, ...
  - File Info : Open file desc table, ...
  - IPC info : signal table, ...
  - Execution context (Depends on OS design)
  - Kernel Stack

- etc.

✦ **CPU Scheduling:**

- CPU scheduler is unit of OS which decides the next process to be executed on the CPU using certain algorithms like FCFS, SJF, RR, etc.

✦ **Memory Management:**

- During compilation, compiler assumes a machine with minimal config (e.g. TC compiler assumes a machine with 8086 CPU and 1 MB RAM); this machine is called as "Virtual Machine".

- Assuming this machine compiler & linker generates addresses for each instruction and data. These addresses are known as "logical addresses" or "virtual addresses".

- During process creation additional virtual addresses are added for runtime section i.e. stack and heap.

- The set of virtual addresses of any process are called as "virtual address space".

- While loading the program into memory, actual locations in the RAM are used, which are called as "physical addresses" or "real addresses".

- The set of physical addresses of any process in the RAM is called as "physical address space".

- CPU always execute a process with its virtual addresses.

- MMU is a hardware unit that converts virtual address of a process (requested by the CPU) into physical address of the process (in the RAM).

- [CPU] -> [MMU] -> [RAM]

- The base and limit of each process is stored in the PCB of that process and will be loaded into the MMU when the process is loaded in CPU (context switch).

- The virtual address requested by the CPU is converted into physical address as shown in diag.

✦ **File & IO Management:**

- File in collection of data or information on secondary storage device.

- File system is used to manage multiple files on the single partition.

- In UNIX/Linux, each device is also treated as file.

✦ **Hardware Abstraction:**

- OS hides the hardware details from the application and end users.

- The changes done in hardware are mostly accepted by the OS without intervention of end users.

✦ **Networking:**

- Data can be transferred from one machine to another machine in some network.

- LAN, MAN, WAN are types of networks.

✦ **Protection & Security:**

- Protecting the system from the internal threats is known as "protection".

- One process cannot access data of another process.

- One user cannot access data of another user.

- Securing the system from external threats like viruses, trojans, worms, etc is known as "security".

- Typical antivirus applications are used to secure the system.

✦ **User Interfacing:**

- User interfacing is optional feature of OS (some embedded OS do not have user interfacing).

- User interfacing is provided by the OS with help of a special program called as "shell".

- This program takes input/commands from the user and get them executed from core OS/kernel. This program is also known as "command interpreter".

- There are two types of user interface/shell:

1. Graphical User Interface:
   - Windows: explorer.exe
2. Command Line Interface:
   - DOS: command.com
   - UNIX/Linux: bsh, bash, csh, ksh, etc.
   - In UNIX/Linux shell runs within terminal program.

✦ **Computer Hardware:**

- CPU executes machine level instruction (in each process).
- Each IO device has its own internal dedicated processing unit called as "IO device controller".
- CPU is connected to the IO devices and memory via "BUS".

✦ **IO Structure:**

- The IO device controller communicates with CPU using special signal called as "Interrupt".
- When interrupt occur. CPU pause current task/process and jumps to IVT.
- In low memory area (1st 1KB of RAM) Interrupt vector table is kept. which contains addresses of interrupt service routines. The ISR contains logic for handling interrupts.
- Then CPU executes appropriate ISR (after getting address from IVT) and handles the interrupt.
- After completion of ISR, CPU resume the pause task/process.

✦ **Types of Interrupts:**

1. Non-maskable Interrupt:

- Interrupts generated from the hardware, which cannot be disabled are called as "NMI".
- Typical error interrupts and other critical interrupts fall in this category.

2. Maskable Interrupt:

- When interrupt arrives, it is possible that CPU is busy in executing some OS critical code. If such code is interrupted, it may cause inconsistency in execution of the OS.
- So during execution of such critical tasks, OS disables interrupts using special assembly language instructions.
- Such interrupts are called as "maskable" interrupts.
- When interrupts are enabled again (after completion of critical task), the arrived interrupts will be visible to CPU and will be handled.
- Such interrupt can generated by hw as well as sw.
- Special software (assembly) instructions can be used to generate software interrupt (TRAP).

  e.g. x86 -> INT instruction

  ARM -> SWI instruction

✦ **Types of IO:**

✦ **Synchronous IO:**

- CPU initiates IO by giving instructions to IO device.
- IO device performs IO and CPU will keep checking whether IO is completed or not. This continuous checking is known as "polling".
- This type of IO is called as "Sync IO" and is easier to implement.
- However in such IO, the CPU is no utilized properly.

✦ **Asynchronous IO:**

- CPU initiate IO by giving instrucion to IO device and continue with execution of some another task.
- When IO device completes IO, it send interrupt to the CPU.
- When interrupt is received CPU can continue to execute the earlier task.
- Such type of IO is called as "Async IO".
- This type of IO increases CPU utilization.
- However in such case, OS need to maintain "Device Status Table".
- This table contains list of all IO devices, their status (busy/idle) and set of processes waiting for each IO device (Waiting queue of IO device).

✦ **Storage Structure:**

- **Memory/Storage:**
  - |- Primary Memory [Directly accessible by CPU] (Volatile)
    - |- CPU registers
    - |- Cache
    - |- RAM (Electronic)
  - |- Secondary Memory [Not accessible to CPU direct] (Non-volatile)
    - |- Hard disk (Magnetic)
    - |- Optical disk (CD/DVD)
- **Data Flow:**
  - CPU Regr <--> Cache <--> RAM <--> Hard disk
- Comparision of storage devices can be done on basis of speed, size and cost.
- **Cache:**
  - Cache is used avoid speed mismatch between CPU and RAM.
  - Data requested by CPU, if not present in cache, it is first copied from RAM to Cache and then from Cache to CPU.
  - If Data requested by CPU, is present in the cache, it is directly taken from there without informing RAM (about data access). This will speed up the data access.
  - If cache is full, the oldest data will be replaced by newer data. Thus cache always contains recent data accessed.

✦ **Booting:**

- If first sector of the storage device/partition contains bootstrap program, then it is called as "bootable device/partition".
- Bootstrap program is different for each OS and can load kernel of that system into the memory.
- Bootloader program displays options for which OS to be started (in case of multiple OS) and depending on user selection starts appropriate bootstrap program.

✦ **Booting Steps:**

- When computer is powered on, programs from the Base ROM (BIOS) are fetched into the RAM automatically.
- The first program from BIOS i.e. POST is executed, which checks whether all devices are functioning properly.
- Then another BIOS program i.e. bootstrap loader is executed, which find the bootable device and starts bootloader program.
- As explained above, bootloader program starts appropriate bootstrap program, which in turn load OS kernel. Thus OS boots.

✦ **Classfication of OS:**

1. Mainframe systems:

✦ **Resident Monitor:**

- Early (oldest) OS resides in memory and monitor execution of the programs. If it fails, error is reported.

✦ **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.
- Thus 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 such 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 use 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 processes (ready for execution) can execute concurrently.
- There are two types of multi-tasking:

1. Process based multitasking: Multiple independent processes are executing concurrently. Sometimes (i.e. on multiple processors) also called as "multi-processing".
2. Thread based multi-tasking OR multi-threading: Multiple parts/functions in a process are executing concurrently.

✦ **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.

E.g. tty, who, who am i, whoami, w, etc.

2. Desktop Systems:

- User convinence and Responsiveness
- Windows, Mac OS X, Linux, etc.

3. 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 "Multi-procssing" programs/OS.
- Since multiple tasks can be exeuted on these processors simultaneously, such systems are also called as "parallel systems".

- There are two types of multiprocessor systems:

A. Asymmetric MP:

- 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.

B. Symmetric MP:

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

4. Distributed Systems:

- Multiple computers connected together in a close network is called as "distributed system".
- It advantages are high availability, high scalability, fault tolerance.
- The requests are redirected to the computer having less load using "load balancing" techniques.
- The set of computers connected together is called as "cluster".

5. Real Time 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 handling of the interrupt is called as "Interrupt Latency". RTOS have very small and fixed interrupt latencies.
- Examples: uC-OS, VxWorks, pSOS, RTLinux, etc.
- Soft real time systems: Not much time critical
- Usually have some secondary storage device.
- Applications: Consumer electronics, ...
- Hard real time systems: Highly time critical
- Usually do not have some secondary storage device.
- Applications: Defence systems, Medical instruments, Space ships, ...

6. Handheld Systems:

- OS installed on handheld devices like mobiles, PDAs, iPODs, etc.
- Challenges:
  - Small screen size
  - Low end processors
  - Less RAM size
  - Battery powered

✦ **OS Design (Implementation):**

1. Simple structure: MS-DOS

```
-----------------------------
COMMAND.COM <- command interpreter
-----------------------------
```

MSDOS.SYS <- kernel

IO.SYS <- device drivers

---

2. Layered Structure:

- OS is divided into multiple layers, so that each layer depends on the lower layer and provide functionality to upper layer.

---

applications

---

system call APIs

---

system call implmenetation

Kernel Executive : File Mgr, Memory Mgr,
Process Mgr, Scheduler, Thread Mgr, etc.

IO Subsystem

Device Drivers

Hardware Abstraction Layer

- Windows, UNIX, etc.

3. Monolithic Kernel:

- Multiple kernel source files are compiled into single kernel binary image. Such kernels are "mono-lithic" kernels.
- Since all functionalities present in single binary image, execution is faster.
- If any functionality fails at runtime, entire kernel may crash.
- Any modification in any component of OS, needs recompilation of the entire OS.
- BSD Unix, Windows (ntoskrnl.exe), Linux (vmlinuz), etc.

4. Microkernel:

- Kernel is having minimal functionalities and remaining functionalities are implemented as independent processes called as "servers".
- e.g. File management is done by a program called as "file server".
- These servers communicate with each other using IPC mechanism (message passing) and hence execution is little slower.
- If any component fails at runtime, only that process is terminated and rest kernel may keep functioning.
- Any modification in any component need to recompile only that component.
- e.g. Symbian, MACH, etc.

5. Modular Kernel:

- Dynamically loadable modules (e.g. .dll / .so files) are loaded into calling process at runtime.
- In modular systems, kernel has minimal functionalities are rest of the functionalities are implemented as dynamically loadable modules.
- These modules get loaded into the kernel whenever they are called.
- As single kernel process is running, no need of IPC for the execution and thus improves performance of the system.
  - e.g. Windows, Linux, etc.

6. Hybrid Kernel:

- Mac OS X kernel is made by combination of two different kernels.
- BSD UNIX ✦ MACH = Darwin

✦ System Calls: [Dual Mode Protection]

- System calls are the functions exposed by the kernel so that user programs can access kernel functionalities.
- Kernel may have number of syscalls. E.g. UNIX OS have 64 syscalls.
- CPU have a mode bit indicating whether user program is running or system program is running.
  mode=0 => kernel/system/privileged mode
  mode=1 => user/non-privileged
- When interrupt occurs, mode is switched from user to kernel mode. And when interrupt is handled (ISR completes) mode is switched back from kernel to user mode.
- System call wrappers/APIs use software interrupt to switch to kernel mode. Then SW intr ISR select appropriate syscall implementation function from System call table and it is invoked.
- In kernel mode the complete access to the system is available and hence all kernel functions/device drivers can directly deal with hardware.
- If any user program try to access hardware directly, CPU will not execute those instructions (mode=1) and such program will be terminated forcefully.
- e.g. open(), close(), write(), read(), lseek(), fork(), exec(), _exit(), ...

✦ CPU Protection:

- To ensure that one process should not consume whole CPU time (e.g. if process stuck in infinite loop), timer hardware is used.
- Timer hardware is configured to generate interrupt periodically, which will cause execution of timer ISR.
- At the end of ISR, CPU scheduler is invoked, which decide the next task to be executed. And control (CPU time) is given to that process.

✦ Process Management:

- Process is program in execution.
- When a process makes system call, functions within kernel are executed. Function activation records of these kernel functions is created on the kernel stack of the process.
- Pointer to kernel stack is maintained in the PCB of the process.

✦ Context Switch:

- If interrupt occured while P1 is executing, values of all CPU registers (execution context) is copied into PCB of the P1 process.

- Then ISR is invoked, which will call scheduler and it will decide the next process to be executed

- Then dispatcher (another unit of OS) will copy the values of CPU registers from the PCB of P2 process into the CPU. Thus CPU will begin/resume execution of P2 process.

- This is called as "context switch".

- During context switch CPU cannot perform any useful task (as CPU registers are getting copied). So very freqent context switch will reduce CPU utilization.

- If context switch is done after long time, then system will not be responsive.

✦ **Process Life Cycle:**

✦ OS Data Structures:

- Job queue / Process table: PCBs of all processes in the system are maintained here.

- Ready queue: PCBs of all processes ready for the CPU execution and kept here.

- Waiting queue: Each IO device is associated with its waiting queue and processes waiting for that IO device will be kept in that queue.

- **Process states:**

  - New, Ready, Running, Waiting, Terminated.

✦ **Inter-process Commmunication:**

✦ Shared Memory:

- A process can transfer data to another process using shared memory.

- It is a common memory area created by OS, in which multiple processes can read/write.

- To keep info about shared memory areas OS maintains structure i.e. shared memory object, which contains: KEY (uniqe id), size, permissions, address of shared memory, number of processes attached to shared memory, etc.

- Addresses of all shared memory objects are kept in a shared memory table and index to shared memory table is called as "shared memory id".

- Imp syscalls:
  - shmget() - create shm region
  - shmat() - get pointer to shm region
  - shmdt() - free pointer to shm region
  - shmctl() - to get info about shm or to delete shm.
  - Commands:
    ipcs -m
    ipcrm -m key
  - Shared memory is fastest IPC mechanism.

✦ **Pipe**

- Pipe is used to communicate betn two processes.
- It is stream based uni-directional communication
- Pipe is internally implemented as a kernel buffer, in which data can be written/read.
- If pipe (buffer) is empty, reading process will be blocked.
- If pipe (buffer) is full, writing process will be blocked.
- If reading process is terminated, writing process will receive SIGPIPE signal.

- There are two types of pipe:

- Unnamed Pipe:

- Used to communicate betn related processes.
  - e.g. who | wc
- Internally pipe is created using pipe() syscall.

- Named pipe / FIFO:

- Used to communicate betn unrelated processes.
- Internally creates an inode and directory entry for a special pipe file on the disk.
- Created using mkfifo command or mkfifo() syscall.
- Opened using open() syscall by reader and writer process.
- write(), read(), close() operations can be performed on both types of pipe.

✦ Message Queue / Mailbox:

- Used to transfer packets of data from one process to another.
- It is bidirectional IPC mechanism.
- Internally OS maintains list of messages called as "message queue" or "mailbox".
- The info about msg que is stored in a object. It contains unique KEY, permissions, message list, number of messages in list, processes waiting for a message to receive (waiting queue).

- IMP Syscalls:

- msgget() - create msg que
- msgctl() - get info or destroy msg que
- msgsnd() - send message into que
- msgrcv() - receive message from que

✦ Signals

- OS have a set of predefined signals, which can be displayed using command

```
kill -l
```

- A process can send signal to another process or OS can send signal to a process.
- kill command is used to send signal to another process, which internally use kill() syscall.
- pkill command is used to send signal to multiple processes/instances of the same program.

```
pkill -SIG programname
```

- IMP Signals:

1. SIGINT: When CTRL✦C is pressed, INT signal is sent to the foreground process. If process does not handle it, process will be terminated.
2. SIGTERM: During system shutdown, OS send this signal to all processes. Process can handle this signal to close resources and get terminated. If process does not handle it, process will be terminated.
3. SIGKILL: During system shutdown, OS send this signal to all processes to forcefully kill them. Any process cannot handle this signal.
4. SIGSTOP: Pressing CTRL✦S, generate this signal which suspend the foreground process.
5. SIGCONT: Pressing CTRL✦Q, generate this signal which resume suspended the process.
6. SIGSEGV: If process access invalid memory address (dangling pointer), OS send this signal to process causing process to get terminated after giving an error message "Segmentation Fault".
7. SIGCHLD: When child process is terminated, this signal is sent to the parent process.

- signal() syscall is used to install signal handler. When signal is received OS calls appropriate signal handler.

✦ **Sockets**

- Socket is defined as communication endpoint.
- Using socket one process can communicate with another process on same machine or different machine in the network.
- Socket = IP address ✦ PORT
- PORT number is 16-bit logical number (0-65535) to identify socket uniquely on a system.
- Two types: TCP sockets / UDP sockets

✦ **Remote Procedure Call**

- ✦ Used to call method from another process on the same machine or different machine in the netwok.
- ✦ Stub and Skeleton are helper objects which transfer data between server and client using sockets.

✦ **Process Creation:**

✦ System Calls:

- Windows : CreateProcess()
- Linux : clone() / fork()
- UNIX : fork()

✦ fork():

- Example:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        int ret;
        printf("start!\n");
        ret = fork();
        printf("return val : %d\n". ret);
        printf("end!\n");
        return 0;
}
```

- 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" and calling process is known as "parent".
- fork() returns 0 to child process and pid of the child process to the parent process.
- 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.
- In UNIX/Linux each process have some parent process.
- The first process (process 1) is "init" process, which is created by a hardcoded "process 0".
- 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.
- Command: ps -e -o pid,ppid,cmd

- When a process terminates, its exit status is returned to the OS. OS store it into PCB of that process (so that its parent can read it).
- 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 (which performs 3 steps):
  - pause execution parent until child process is terminated.
  - read exit status & other info from PCB of child process & return to parent process (as out param).
  - release PCB of the child process.

✦ **exec() syscall:**

- exec() syscall "loads a new program" in the calling process's memory (address space) and replaces the older one.
- There are six functions in the family of exec():
  - execl(), execlp(), execle(),
  - execv(), execvp(), execve()
- Example:

```
ret = fork();
if(ret==0)
{
        err = execl("/bin/ls", "ls", ..., NULL);
        if(err < 0)
        {
                perror("execl() failed");
                _exit(1);
        }
}
wait(&staus);
```

- If exec() succeed, it does not return (rather new program is executed).

✦ **Program/Command Execution:**

✦ Synchronous Execution: Parent waits for the child process to complete its execution. Internally calls wait function. By default shell executes each command synchronously.

e.g. ./kcalc

✦ Asynchronous Execution: Parent does not wait for the child process to complete its execution. Shell can execute a command asynchronously by suffixing & to the command.

e.g. ./kcalc &

✦ **CPU Scheduling:**

✦ There are four scenarios when CPU scheduler is invoked:

A. Running -> Terminated (exit)
B. Running -> Waiting (io request)
C. Running -> Ready (interrupt)
D. Waiting -> Ready (io completes)

✦ **Two Types of scheduling:**

1. **Non-Preemptive Scheduling:**
   - Scheduler is invoked only in case A & B (where process give up CPU on its own).
   - Also called as "co-operative scheduling".

2. **Preemptive Scheduling:** Scheduler is invoked in all above cases (Specially in case C & D, running process is forced to leave the CPU).

✦ **CPU Scheduling Criterias:**
- CPU Utilization: MAX : Server:90%, Workstation:60%
- Throughput: MAX : Amount of work completed in unit time.
- Waiting Time: MIN : Time spent by the process in the ready queue waiting for the CPU.
- Turn around Time: MIN : Time from the arrival of process to termination of the process.
- Response Time: MIN : Time from the arrival of the process to first CPU allocation time for that process.

✦ **CPU Scheduling Algos:**

✦ FCFS
- Process arrived first will be scheduled first.
- Non-preemptive scheduling
- Avg waiting time depends on the order of arrival of the process.

✦ SJF
- Process having minimum burst time will be executed first.
- This algo gives minimum waiting time.
- Can be implemented as Non-premptive as well as Preemptive
- Preemptive SJF is also known as "Shortest Remaining Time First".

✦ **Priority**
- Each process is associated with a number called as priority of the process. Lower is the number, higher is the priority.
- Can be implemented as Non-premptive as well as Preemptive
- Process having lower priority may not get sufficient CPU timer for execution; this is called as "Starvation".
- To resolve this priotity of such processes can be incremented periodically; this is called as "Aging".

✦ **Round Robin**
- Each process is given a small CPU time (called as time quantum) repeatedly.
- Pre-emptive algorithm
- This algo gives minimum response time.

✦ **Fair Share:**
- CPU time is divided into epoch times.
- Each process will get fair amount of CPU time in each epoch based on its priority [nice value].
- By default a process inherit nice value from the parent process. It can be altered programmatically

using nice() system call.

- Root user can change priority of the process using "nice" or "renice" command.
- Lower is the nice value, higher is the CPU time share given to the process.

✦ **RTOS Scheduling Algorithms:**

✦ **Rate Monotonic Scheduling:**

- Static priority based scheduling algorithm i.e. priorities of the processes are not changed at run time.
- Does not utilize CPU fully as number of processes increase.

✦ **Earliest Deadline First:**

- The priorities of processes are dynamically changed to complete the processes within deadline.

✦ **Proportional Share Scheduling:**

- Based on priority of the process, it is given some amount of CPU shares.

✦ **Multi-Level Queue:**

- Depending on nature of the processes, the ready queue is splitted into multiple sub-queues and each queue can have different scheduling algo. This is known as "multi-level queue".
- If a process is not getting sufficient CPU time, in its current queue, then it can be shifted into another queue by OS. This enhanced concept is known as "multi-level feedback queue".

✦ **Process 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.
- To resolve this problem, one process can access resource at a time. This can be done using sync objects/primitives given by OS.
- Well known sync objects are:
  - Semaphore, Mutex, Event, Critical section, Waitable Timer, Monitor

✦ **Semaphore:**

- Semaphore is a sync primitive given by OS.
- Internally semaphore is a counter. On semaphore two operations are supported:

1. wait operation: dec op: P :

- semaphore count is decremented by 1.
- if cnt < 0, then calling process is blocked.
- typically wait operation is performed before accessing the resource.

2. signal operation: inc op: V :

- 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.

Q. If sema count = -n, how many processes are waiting on that semaphore?

Ans: "n" processes waiting

Q. If sema count = 5 and 3 P & 4 V operations are performed, then what will be final count of sema?

Ans: 5 - 3 ✦ 4 = 6

✦ There are two types of semaphores:

1. Counting Semaphore:

- Allows "n" number of processes to access resource at a time.

2. Binary Semaphore:

- Allows only 1 process to access resource at a time.

✦ **IMP Syscalls:**

- sem_init() : init sem with some count
- sem_destroy() : destroy sem
- sem_wait() : wait/dec op
- sem_post() : signal/inc op

✦ **Mutex:**

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is same as "binary semaphore".
- However mutex is more efficient than binary semaphore.
- Mutex can be unlocked by the same process/thread, which had locked it.
- IMP syscalls:
  - pthread_mutex_create() : create mutex
  - pthread_mutex_destroy() : destroy mutex
  - pthread_mutex_lock() : lock mutex
  - pthread_mutex_unlock() : unlock mutex

✦ **Deadlock**

✦ Deadlock occurs when four conditions/characteristics hold true at the same time:

1. No preemption: A resource should not be released until task is completed.
2. Mutual exclusion: Resources is not sharable.
3. Hold & Wait : Process holds a resource and wait for another resource.
4. 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.

✦ **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 :**

1. 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.

2. Banker's algorithm: A bank always manage its cash so that they can satisfy all customers.

3. 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.

Q. A system has 3 processes sharing 4 resourses. If each process needs a maximum of 2 units, deadlock ____

Ans:

3 * 2 < 4 ✦ 3

6 < 7 -> TRUE => Deadlock will not occur

✦ Multi-Threading:

- Multiple parts/functions in a process can be executed concurrently using "threads". It is called as "thread based multi-tasking" or "multi-threading".
- When OS creates a new thread, a new stack is created for the thread within process memory and also a controlling block is created for the thread (TCB).
- Each thread is associated with a function called as "thread function" or "thread procedure". When thread is created, it begins execution of that function and when function completes, thread is terminated.
- TCB of a thread contains info required to execute thread, which includes:
  - Thread Id, Scheduling Info (state, sched algo), User Stack address, Execution context, Kernel stack, etc.
- In modern OS, process is like a container and threads are executed within processes. They are executable units.
- In modern OS, scheduler allocated CPU time to the threads (not the process).
- In modern OS, for each process at least one thread is created by the OS, which execute the main() function. This thread is called as "main" thread.
- Programmer can create additional threads using syscalls or library functions:
  - Windows: CreateThread( )
  - Linux: clone()
- It is standard programming to create threads using libraries instead of syscalls.
- POSIX thread library is used to create thread on UNIX/Linux/Mac OS X.

```
pthread_t th;
ret = pthread_create(&th, NULL, func, NULL);
```

arg1: out param contains thread id.
arg2: thread attributes e.g. stack size, priority, scheduling policy
arg3: address of thread func void* func(void*);
arg4: argument of the thread func.

- Thread is a lightweight process.
  - When a thread is created a new TCB and stack is created; while other sections are shared from the parent process.
  - Threads can communicate with each other directly via global/dynamic variables (data/heap section). Their communication is much faster than IPC.
  - Context switch between two threads of the same process is much faster.
- If parent process is terminated (container), all threads in that process are terminated.

✦ **Threading Model:**

- Threads created by thread libraries are used to execute functions in user program. They are called as "user threads".
- Threads created by the syscalls (or internally into the kernel) are scheduled by kernel scheduler. They are called as "kernel threads".
- User threads are dependent on the kernel threads. Their dependency/relation (managed by thread library) is called as "threading model".
- There are four threading models:

1. Many to One:

- Many user threads depends on single kernel thread.
- If one of the user thread is blocked, remaining user threads cannot function.

2. Many to Many:

- Many user threads depend on equal or less number of kernel threads.
- If one of the user thread is blocked, other user thread keep executing (based on remaining kernel threads).

3. One To One:

- One user thread depends on one kernel thread.

4. Two Level Model:

- OS/Thread library supports both one to one and many to many model.

✦ **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 mgmt schemes are available:
  1. Contiguous Allocation:
  2. Segmentation:
  3. Paging:

✦ **Contiguous Allocation:**

A. 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".

B. 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.

1. First Fit: Allocate first free slot which can accommodate the process.
2. Best Fit: Allocate that free slot to the process in which minimum free space will remain.

3. Worst Fit: Allocate that free slot to the process in whic 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

✦ Memory Management (continued):

✦ 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 appripriate base-limit pair from MMU is used to calculate physical address as shown in diag.

- MMU also contains STBR register which contains address of 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 seg 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.

✦ **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.
- **Context switch can be handled in one of the following ways:**

1. During context switch TLB is flushed i.e. all entries used by previous process will be cleared. Thus new entries page addresses will not conflict with prev process page addresses.

2. TLB can store page addresses along with PID of the process. In this case there is no need to flush TLB during context switch. The comparison of the page address will not conflict with another process, because it is associated with PID of the process. This combination PID and Page Address is called as "ASID".

✦ **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

- If frame size is 4KB, 12 bits are sufficient to speficy any offset in the frame. This will also ensure that "d" will not contain any invalid frame offset.

- If virtual address of a process is of 32 bits [p1|p2|d], then maximum address of the process can be
= 1024 * 1024 * 4 * 1024 = 4 GB.

✦ **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 swapped out.
- 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 program.
2. If virtual address is valid (i.e. page is swapped out), then locate one empty frame in the RAM.
3. If page is on swap device, swap in the page in that frame.
4. Update page table entry i.e. add new frame address and valid bit = 1.
5. Restart the instruction for which page fault occurred.

✦ **Page Replacement Algorithms:**

- While handling page fault if no empty frame found (step 2), 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 algos:

1. 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".

2. 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.

3. 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".

✦ **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 system performance is reduced.
- The problem can be solved by increasing RAM size in the machine.

✦ **Copy On Write:**

- fork() syscall creates a logical copy of the calling process.
- Initially, child and parent both processes share the same pages in the memory pages.
- When one of the process try to modify contents of a page, the page is copied first: so that parent and child both will have seperate physical copies of that page. This will avoid modification of a process by another process.
- This concept is known as "copy on write".
- The primary advantage of this mechanism is to speed up process creation (fork()).

✦ **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.

✦ **Demand Paging:**

- When virtual memory is used with paging memory management, pages can be swapped out in case of low memory.
- These pages will be loaded into main memory, when they are requested by the CPU. This is called as "demand paging".

✦ **File Management:**

- File is collection of data/info on secondary storage device.
- Any file has
  - data (contents of the file)
  - meta-data (info about the file).
    - size of file
    - permissions of file
    - user/group info
    - time stamps : creation, accessed, modified
    - type of file
    - info about data blocks of that file.

✦ **Hard Disk:**

- Hard disk is a block device. The data can be read/written in the disk block by block.
- Number of bytes that be read/written on the disk at a time is called as "physical block size" (sector). Typically it is 512 bytes and is decided by the manufacturer.
- OS/Filesystem may read/write disk in terms of larger block size, which is called as "logical block size".
- Logical block size is always multiple of physical block size.

✦ **File System:**

- File system is way of organizing the contents on the disk.
- During formatting file system is created.

- Formatting commands:
  - Windows : format
  - UNIX/Linux : mkfs
- Any file system is logically divided into four parts:
  - Boot block / Boot sector:
    - Contains bootstrap and boot loader program is partition is bootable.
  - Super block / Volume Control block:
    - Contains info about the partition
    - Partition size, label
    - Number of data blocks, number of free data blocks and info about free data blocks.
  - i-node list / Mater file table:
    - The info/metadata of a file is stored in a structure called as "file control block" or "i-node".
    - Collection i-nodes in a partition is called as "i-node list".
  - Data blocks:
    - The data/contents of a file are stored in data blocks.

✦ **Types of File:**

- From (UNIX) OS perspective there are 7 types of files:
  - regular file
  - d directory file
  - l link file
  - p pipe/fifo file
  - s socket file
  - c char device file
  - b block device file

✦ **Directory:**

- From end user perspective, directory is a container which contains sub-directories and files.
- However, OS treats directory as a special file. The directory file contains one entry for each subdirectory or file in it.
- Each directory entry contains i-node number and name of sub-dir / file.

✦ **Links:**

- There are two types of links in Linux/UNIX.

1. Symbolic Link:

- ln -s /path/of/target/file linkpath
- Internally use symlink() syscall.
- A new link file is created (new inode and new data block is allocated), which contains info about the target file.
  - Link count is not incremented.
    - If target file is deleted, link becomes useless.
    - Can create symlinks for directories.

2. Hard Link:

- ln targetfilepath linkpath
- Internally use link() syscall.
- A new directory entry is created, which has a new name and same inode number. No new file (inode and data blocks) is created.
- Link count in the inode of the file is incremented.
  - If directory entry of target file is deleted (rm command), file can be still accessed by link directory entry.
  - Cannot create link for directories.

✦ **rm command:**

- The rm command in Linux, internally calls unlink() system call.
- It deletes directory entry of the file.
- It decrements link count in the inode by 1.
- If link count = 0, the inode is considered to be deleted/free. It can be reused for any new file.
- When inode is marked free, data blocks are also made free, so that they can also be reused for some new file.

✦ **File System Architecture:**

- Virtual File System:
  - This layer redirect file system request to the appropriate file system manager.
- File system manager:
  - File system manager enables access to repective file system on the disk.
  - OS can see all partitions whose file system managers are installed in that OS.
- IO subsystem:
  - Implement buffer cache and other mechanisms to speed up disk IO.

✦ **open() syscall:**

fd = open("/home/nilesh/abc.txt", O_RDONLY);

1. Convert given file path into its inode number. This is called as path name translation and is done by a kernel internal function namei().

2. Read inode of the file from the disk into inode table in memory. Inodes of all recently accessed files are kept in this table.

3. A file position is initialized to 0 and is stored in the open file table. Info of all files opened in the system, is maintained in this table.

4. Each process is associated with a file descriptor table. It keeps info of all files open by that process. For any process, three files are opened by default: 0-stdin, 1-stdout, 2-stderr. This entry stores mode in which file is opened and pointer to OFT entry.

5. Finally index to file desc table entry is returned, which is called as "file descriptor". All further read(), write(), lseek(), close() operations will be using this file desc.

✦ **Disk Allocation Mechanisms:**

Each file system allocate data blocks to the file in different ways:

1. Contiguous Allocation:

- Number of blocks required for the file are allocated contiguously.
- inode of the file contains starting block address and number of data blocks.

- Faster sequential and random access

- Number of blocks required for the file may not be available contiguously. This is called as "External Fragmentation".

- To solve this problem, data blocks of the files can be shifted/moved so that max contiguous free space will be available. This is called as "defragmentation".

2. Linked Allocation:

- Each data block of the file contains data/contents and address of next data block of that file.
- inode contains address of starting and ending data block.
- No external fragmentation. faster sequential access
- Slower random access.
- e.g. FAT

3. Indexed Allocation:

- A special data block contain addresses of data blocks of the file. This block is called as "index block".
- The address of index block is stored in the inode of the file.
- No external fragmentation. faster random and sequential access.
- File cannot grow beyond certain limit.

✦ **Free Space Management Mechanisms:**

- To allocate free blocks to any file, they should be located quickly. For this reason, info of free blocks is maintained in the super block using one of the following algorithm:

1. Bit Vector:

- Array of bits is used to keep info of used/free blocks.
- Number of bits = number of blocks.
- If nth bit=1, it means nth block is used.
- If nth bit=0, it means nth block is free.

2. Linked List:

- Super block contains address of first free block.
- Each free block contains address of next free block.

3. Grouping:

- One free block contains addresses of remaining free blocks.

4. Counting:

- Each entry in a free block keep starting free block address and number of free blocks after that.

✦ **Disk Scheduling:**

✦ **Hard disk structure:**

- Time required to perform read/write operation on particular sector of the disk, is called as "disk access time".
- Disk access time includes two components = seek time and rotational latency.
- Seek time is time required to move head to desired cylinder (track).
- Rotational latency is time required to rotate the platters so that desired sector is reached to the head.

✦ **Disk Scheduling Algorithms:**

- When number of requests are pending for accessing disk cylinders, magnetic head is moved using certain algo. They are called as "disk scheduling algorithms".

1. FCFS:

- Requests are handled in the order in which they arrived.

2. SSTF - Shortest Seek Time First:

- Request of nearest (to current position of magnetic head) cylinder is handled first.

3. SCAN or Elevator:

- Magnetic head keep moving from 0 to max cylinder and in reverse order continuously serving cylinder requests.

4. C-SCAN:

- Magnetic head keep moving from 0 to max cyliner serving the requests and then jump back to 0 directly.

5. LOOK:

- Implementation policy of SCAN or C-SCAN.
- If no requests pending magnetic head is stopped.

****

✦ **Linux File System:**

- Hierarchy:

/ (root of filesystem)

|- bin (programs and commands)
|- sbin (system programs and commands)
|- lib (library files used by various applns *.so)
|- boot (bootloader, kernel image & booting files)
|- home (for each user one directory is created here).
|- root (home dir of root user)
|- usr (user installed programs)
|- var (for system logs)
|- dev (files representing hw devices)
|- proc (info about the system)
|- sys (info about the devices)
|- etc (system config files)

- If username is "dac", then its home directory is "/home/dac". Any user must save all data/files into his/her home directory.
- The "root" user in Linux have full access to all system files. The "root" user's home directory is "/root".
- config files:
  - /etc/passwd (username and passwd info)
  - /etc/inittab (username and shell info)
  - /etc/fstab (info of mounted partitions)
- proc filesystem:
  - /proc/cpuinfo : complete hardware detail of CPU.

✦ **Linux Commands:**

- Linux is a multi-user system i.e. multiple users can login and work concurrently on the single Linux machine.
- If Linux is installed on your machine [SuSE Linux], then the terminals can be accessed using keys as follows:

  CTRL ✦ ALT ✦ F1 -> tty1

  to

  CTRL ✦ ALT ✦ F7 -> tty7 -> :0 -> GUI
- Within GUI terminal, multiple terminal windows can be opened, called as psuedo terminals. e.g. pts/0, pts/1, ...
- The current terminal name can be found using "tty" command.
- Command for knowing current user name: whoami
- Command for knowing current user, current terminal and login time is: who am i
- To know all users connected to the server machine: who
- To know detailed info about uses connected to server: w
- To close the terminal: exit

✦ **Path:**

✦ **absolute path:**

- full path of the file or directory
- always starts from "/" (root of file system)

✦ **relative path:**

- path of file or directory with respect to current directory
- does not start with "/"

✦ **Special Directory Symbols:**

.. -> represent parent directory

. -> represent current directory

~ -> represent user's home directory

✦ **Linux Basic Commands:**

1. pwd
2. mkdir dirpath
3. ls

   ls dirpath

   ls -l
4. cd dirpath
5. rmdir dirpath
6. cat > filepath

   ....

   ....

   CTRL + D
7. cat filepath
8. rm filepath
9. rm -r dirpath
10. cp command:

    cp filepath dirpath => copy given file in given dir

cp filepath destfilepath=> copy given file with another name

cp -r dirpath destdirpath=> copy given dir to given dest dir

11. mv command:

mv filepath dirpath => move given file into given dir

mv dirpath destdirpath => move given dir into given dest dir

mv filename newfilename => rename a file.

12. man commandname

13. touch filepath

- If file already exists, its "modification time" will be changed.
- If file does not exists, a new empty file is created.

14. echo command:

echo "to display a message"

echo -e "hi\tall\nhow was 1st day with linux?\ngood??"

echo -n "no newline after the text"

echo "can display variable value using $varname"

15. Environment variables keep important info about the system.

Command: env

16. grep -> GNU Reguar Expression Parser

- Used to search a pattern in single/multiple file(s).
- grep "word" filepath
- grep -R "word" dirpath
- egrep "fibonac+i" filepath

+ -> char is repeated 1 or more times

* -> char is repeated 0 or more times

? -> char is repeated 0 or 1 time

{n} -> char is repeated n times

{m,n} -> char is repeated minumum m and maximum n times

^ -> match at beginning of line

$ -> match at the end of line

[char1|char2] -> search a from options

[c1-c2] -> search a char from given range

. -> any one character

grep -> to search basic regular expression

egrep -> to search extended regular expression -> grep -E

fgrep -> to search a fixed word -> grep -F

17. chmod command:

- Used to give permissions to a file.
- chmod +x filepath
  - will add execute permission to all [ugo].
  - +x, +w, +r to add execute, write and read perms respectively.
- chmod -w filepath
  - will remove write permission from all [ugo].

- -x, -w, -r to remove execute, write and read perms respectively.

- chmod u+x filepath
  - will add execute permission for the user (owner of file)
- chmod g-w filepath
  - will remove write permission for the group.
- u+w, u+r, u+x, u-w, u-r, u-x
  g+w, g+r, g+x, g-w, g-r, g-x
  o+w, o+r, o+x, o-w, o-r, o-x
- to give permission to file octal form
  PERM -> rwx r-x r--
  111 101 100
  7 5 4
  chmod 754 filepath
- to change permission of all files in a directory
  chmod -R perm dirpath

18. chown command:
    - chown newuser filepath
    - chown newuser:group filepath
    - chown -R newuser:group dirpath

✦ **Redirection:**

1. output redirection
   command > filepath
   e.g. ls -l /home > output.txt
   command >> filepath
2. input redirection
   command < filepath
   e.g. wc < input.txt
   e.g. wc < input.txt > output.txt
3. error redirection
   command 2> filepath
   e.g. ls -l /ho 2> error.txt
   * command < input.txt > output.txt 2> error.txt

✦ **Pipe**

- Used to communicate between two processes.
- Output of a command can be redirected to another command.
- command1 | command2
- e.g. cat -n names.txt | head -11 | tail -7

✦ **VI editor**

- Developed by Bill Joy

- Editor works in two modes
  - Insert (Edit) mode : Press "i"
  - Command mode : Press "Esc"
- To open/create a file using VI editor
  vim filepath
- VI editor basic commands
  :w -> write
  :q -> quit
  :wq -> write and quit
  :q! -> quit without saving
- VI editor advanced command (to customize VI)
  :set number
  :set autoindent
  :set tabstop=4
  :set shiftwidth=4
  :set autowriteall
- VI editor copy/paste related commands:
  yy -> copy current line
  n yy -> copy n lines from cursor
  :m,ny -> copy from mth line to nth line
  p -> paste
  dd -> cut current line
  n dd -> cut n lines from cursor
  :m,nd -> cut from mth line to nth line
  u -> undo
  CTRL R -> redo
  y$ -> copy from cursor to end of line
  y^ -> copy from cursor to start of line
  yw -> copy current word
  n yw -> copy n words after the cursor
  d$ -> cut from cursor to end of line
  d^ -> cut from cursor to start of line
  dw -> cut current word
  n dw -> cut n words after the cursor

✦ **C Programming on Linux:**

step 1: vim filename.c
  Write and save the C source code.
step 2: gcc filename.c -o filename.out
  Compile .c file and create executable (.out) file.
step 3: ./filename.out
  To run the program.

✦ **C++ programming on Linux:**

step 1: vim filename.cpp

Write and save the C++ source code.

step 2: g++ filename.cpp -o filename.out

Compile .cpp file and create executable (.out) file.

step 3: ./filename.out

To run the program.

✦ **C/C++ Program Debugging:**

- While compilation use -g flag

gcc filename.c -g -o filename.out

- While debugging

gdb ./filename.out

- Set break points wherever appropriate

gdb> break funname

gdb> break linenum

gdb> run

- Important gdb commands:

cont -> run till next break point

step -> step into the function

next -> step over the function

print varname -> watch variable value

quit -> stop debugging

help -> to see help about gdb commands

✦ **BASH Shell Scripts:**

- BASH reference manual - ebook

- #!/bin/bash -> shebang line
- comments starts with #

- do not use $ sign while assigning value to var

var=123

- to read value of var use $ sign

echo $var

newvar=$var

- to execute command and assign output to a variable

var=$(command ...)

var=`command ...`

- to execute command

command ...

- relational operator

-eq -ne -lt -le -gt -ge

- logical operators

-a -o !

- if [condition]
  then
  ...
  fi

- if [condition]
  then
  ....
  else
  ....
  fi

- Can use "elif" as combination as else-if.

- case expression in
  c1)
  ...
  ;;
  c2)
  ...
  ;;
  *)
  ...
  esac

- while [condition]
  do
  ....
  done

- break & continue can be used with all types of loops (while, until, for loop) like C/C++

- until [condition]
  do
  ....
  done

- for var in collection
  do
  ....
  done
  The for loop is used to access elements one by one from the collection.

- Special Operators to use in a condition.
  -e check if path exists
  -d check if path is of directory

-f check if path is of file
-r check if path is readable
-w check if path is writable
-x check if path is executable

- All conditions in script [...] are executed internally with "test" command.
- man test

✦ **Positional Parameters:**

- Extra info passed to the script while executing it on command line is called as "positional parameters".
- ./scriptname arg1 arg2 arg3 arg4
- They can be accessed in script as : $1, $2, ..., $9
- Number of positional parameters : $#
- List of positional parameters : $*
- Name of current shell script : $0
- "shift n" command discard first n args and next arguments can be accessed
- Example:

```
#!/bin/bash
echo $1
echo $2
shift 2
echo $1
echo $2
```

- If above script runs as: ./script arg1 arg2 arg3 arg4

The output will be:

```
arg1
arg2
arg3
arg4
```

- Using shift command we can access parameter 10 and onwards.

- Functions:

```
function funname()
{
        # args are accessed as $1, $2, ...
        ....
}
res=$(funname ...)
```

- Functions must be defined at the start of the script
- All results of echo statements will be kept in temp string and will be returned at the end of the function, which can be collected in another variable as shown above.

Q.1 An OS uses a paging system with 1K pages. A given process uses a virtual address space of 128K and is assigned 16K of physical memory. How many entries does its page table contain?

Answer: 128

Explain:

virtual address space =	128KB	131072	Bytes
number of frames/pages =	1K	1024	
size of one page =	virtual addr space / no of pages	128	Bytes
physical memory of process =	16KB	16384	Bytes
number of pages for process =	physical memory / page size	128	
number of page table entries =	number of pages for process	128	

Q.2 An OS uses the elevator algorithm to schedule the disk-arm. I/O requests are currently pending for blocks on tracks 1, 3, 8, 11, 15, and 16. The disk arm is currently at track 9 and moving upwards. In what order will these requests be handled?

Answer: 11, 15, 16, 8, 3, 1

Explain: elevator algorithm = SCAN algorithm

Q.3 A particular system uses a page size of 1K bytes. A page table for a particular process begins as follows: [3, 4, *, 1, *, 8 ...]
* A. What physical address corresponds to the virtual address of 50?
* B. Name a virtual address that will generate a page fault.

Answer: A: 3122
B: Any Addr betn 2048 to 3071 OR betn 4096 to 5119

Explain:

Logical Address	Physical Address	Page Table: Logical Page	Physical Frame	Valid/Invalid
0	3072	0	3	valid
1024	4096	1	4	valid
2048	INVALID	2	*	invalid
3072	1024	3	1	valid
4096	INVALID	4	*	invalid
5120	8192	5	8	valid

A: Address 50 = Page 0 address => Physical Frame 3 i.e. Frame 3 address + 50 (offset) => 3072 + 50 = 3122
B: Range of addresses for which physical addresses are not present : Any virtual address between 2048 and 3071 or between 4096 and 5119

Q.4 **With a segmentation, if there are 64 Segments and the maximum segment size is 512 words, the length of the logical address in bits is_____**

Answer: **16**

Explain: In segmentation scheme, Logical address contains "segment address" and "offset address".

64 segments :	segment address bits =	6
512 words = 512 * 2 = 1024 bytes :	offset address bits =	10
	Total Address Bits =	16

Q.5 **In an operating system using paging , if each 32-bit address is viewed as a 20-bit page identifier plus aS 12 bit offset, what is the size of each page?**

Answer: **4KB**

Explain: Page offset = 12 bits

So Page size =	2 ^ 12	4096 bytes

Q.6 **A 1000 Kbytes memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 kbytes and 260 Kbytes respectively. The largest allocation request in Kbytes that could be denied is for _____.**

Answer: **541 KB**

Explain: Assuming that both allocations are done contiguously at the bottom or top of the memory, the rest of the memory will be empty

Total Memory :		1000 KB
Total Allocation:	200 KB + 260 KB =	460 KB
Remaining Free Memory :	1000 KB - 460 KB =	540 KB

So cannot allocate the block greater than it...

Q.7 **A 1000 Kbytes memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 kbytes and 260 Kbytes respectively. The smallest allocation request in Kbytes that could be denied is for _____.**

Answer: **181 KB**

Explain: For the smallest allocation of the memory, we need to assume that memory is allocated in the middle as follows so that X, Y, Z almost equal ...

Total Memory :		1000 KB
Total Allocation:	200 KB + 260 KB =	460 KB
Remaining Free Memory :	1000 KB - 460 KB =	540 KB

Divide remaining memory so that X, Y, Z will be equal i.e. Remaining Memory Divided by 3

Smallest Available Slot =	540 KB / 3	180 KB

So we cannot allocate any slot larger than it

X
200
Y
260
Z

Q.8 If Semaphore values have 12, then 6V and 4P operations the resultant value is _____

Answer: 14

Explain: Each P operation decrements semaphore count by 1
Each V operation increments semaphore count by 1
Original Count = 12
Incrementing 6 Times = 18
Decrementing 4 Times = 14
Final Count = 14

Q.9 **Disk request come to a disk driver for cylinders in the order 10,22,20,2,40,6 and 38, at time when the disk drive is reading from cylinder 20. The seek time is 6 ms per cylinder. the total seek time, if the disk arm sheduling algorithm is first come first served is _____.**

Answer: **876 ms**

Explain: Seek Sequence for FCFS will be :
20, 10, 22, 20, 2, 40, 6, 38

Total Seek Distance =	146
Seek time per cylinder =	6 ms
Total Seek Time =	876 ms

20	10
10	12
22	2
20	18
2	38
40	34
6	32
38	
Total Seek Dist :	146

Q.10 Each process p(i)=1,2,3,.....,9 is coded as follows
repeat
p(mutex)
{critical section}
v{mutex}
forever
The code for p10 is identical except that it uses V(mutex) instated of p(mutex).What is the largest no of processes that can be the inside the critical section at any moment?

Answer: 3

Default behaviour of P() / wait operation locks mutex, while V() / signal operation unlocks mutex.
Due to this nature, only one process out of P1 to P9, will be inside the mutex.(Now there is 1 process)
The P10 process calls V() operation and enters into critical section (Now threre are 2 processes)
However, since P10 unlock the mutex, one more process from P1 to P9 can enter into critical section. (Now threre are 2 processes)
Thus there can be at max 3 processes in the critical section.

Q.11 **At a perticular time of computation the value of counting semaphore is 7. Then 20 `p` operation and x `v` operation were completed on this semaphore. If the final value of the semaphore is 5, x will be _____**

Answer : 18

P() operation decrement count, while V() operation increment the count.

Current Count =	7
P() Operations =	20
New Count =	-13
Final Count =	5
V Operations =	18

Q.12 **A semaphore count of negative n means (s=-n) that the queue contains _____ waiting processes.**

Answer : n

If semaphore count is zero and any process perform decrement operation the process goes to waiting state,
Thus the first process doing decrement operation will be waiting and semaphore count -1.
So n waiting processes will make semaphore count = -n

Q.13 **The OS uses a round robin scheduler. The FIFO queue of ready processes holds three processes A,B,C in that order. The time quantum is 18 msec. A context switch takes 2 msec. After running for 13 msec, B will block to do a disk read, which will take 30 msec to complete. Trace what will happen over the first 100 msec. What is the CPU efficiency over the first 100 msec?**

Answer : 90%

A FIFO Queue indicate that processes are added at the end of the queue and removed from the beginning (it is not FCFS algo).
During context switch CPU is not doing some fruitful work and hence is not utilized at that time.
When a process goes for IO, it is removed from READY queue and is added to IO WAITING queue of particular device.
Considering all these points the seqence of scheduling will be as follows (including context switch):
A (18) => SW (2) => B (13 io) => SW (2) => C (18) => SW (2) => A (18) => SW (2) => C (18) => SW (2) => B (5)

Total Milliseconds =	100 ms
Unused time (switching) =	10 ms
CPU utilization / efficiency =	90 ms
CPU utilization =	90 %

Prasanna Wadekar

Q.14 In a page memory the page hit ratio is 0.35. The time required to access the page in secondary memory is equal to 100 nanosec. the time required to access page in primary memory is 10 ns . the avg time reqd to access page is _____.

Answer : 68.5 ns

Page hit ratio represent probability of getting the page in main memory.

Hit ratio = 0.35 i.e. out of 100, only 35 pages will be available in main memory.

i.e. out of 100, only 65 pages will be available taken from secondary memory.

Time for accessing page in memory = 10 ns

Total time for accessing 35 pages = 350 ns

Time for accessing page in disk = 100 ns

Total time for accessing 65 pages = 6500 ns

Total time for accessing 100 pages = 6850 ns

Average time for accessing 1 page = 68.5 ns

Q.15 A 1000 MB hard disk has 512-byte sectors. Each track on the disk has 1000 sectors. The number of tracks on the disk is _____.

Answer : 2048

disk size = number of tracks * sectors per track * sizeof one sector

number of tracks = disk size / (sectors per track * sizeof one sector)

number of tracks = 1000 MB / (1000 * 512)

2048

Q.16 **The address sequence generated by tracing a particular program executing in a pure demand paging system with 100 records per page, with 1 free main memory frame is recorded as follows. what is the number of page faults?**
0100, 0200, 0430, 0499, 0510, 0530,0560, 0120, 0220, 0240, 0260, 0320,0370

Answer : **7**

Here no of records per page is 100, so the page no corresponding to these address will be
1,2,4,4,5,5,5,1,2,2,2,3,3
Also only one frame is available along with pure demand paging
So, Page fault will occur for frame nos. 1,2,4,5,1,2,3
Hence no of page fault is =7

Q.17 **Peak bandwidth of 64 bits, 33 MHZ based PCI bus would be _____**

Answer : **266 MB / sec**

Peak Bandwidth =	Bus Width (in Bytes) * Bus Freq (in Hz)	
Bus Width =	64 Bits	8 Bytes
Bus Freqnecy =	33 MHz	33.3 MHz
Peak Bandwidth =		266.4 MB/sec

Q.18 **Cosider a system having m resourses of same type. The resourse are shared by 3 processes A,B,C which have peak time demands of 3,4,6 respectively. The minimum value of m that ensures that deadlock will never occur is _____.**

Answer : **11**

Formula for resources of the same type :
Total of max needs < no. of resource instances + no. of processes
13 < m + 3
So, m >= 11

Q.19 **A certain moving arm disk storage with one head has following specification:**
Number of tracks/recording surface = 200
Disk rotation speed = 2400 rpm
Track storage capacity = 62500bits
the avg latency time (assume that the head can move from one track to another only by traversing the entire block)

Answer : **2.5 sec**

Number of tracks =	200	
Number of rev per sec =	2400 / 60	40
Time required for 1 revolution =	1 / 40	0.025 sec
Correct Average Latency =	0.5 * time for 1 rev	0.0125 sec
But Ans Average rotational latency =	0.5 * tracks * time for 1 rev	2.5 sec

Track storage capacity =	62500 bits	
Data transfer rate =	number of rev (tracks) per sec * track storage capacity	
	i.e. 40 * 62500 bits / sec	
	2500000 bits/sec	
	2.5 mb/sec	

Q.20 A system has 3 processes sharing 4 resourses. If each process needs a maximum of 2 units, deadlock ____
Answer: Can Never Occur

Formula for resources of the same type :
Total of max needs < no. of resource instances + no. of processes
6 < 4 + 3
This holds true, indicating that deadlock never occur

Q.21 In a multi-user OS 20 requests are made to particular resource per hour on an avg, the probability that no request are made in 45 minute is ___
Answer: e ^ -20

Avg requests per unit time (1 hr) = 20
Num of req per 45 mins i.e. L = 15

Probability is given by = (L ^ x * e ^ -L) / x!
x is num of expected reqs i.e. 0
So probability will be = (15 ^ 0 * e ^ -15) / 0! = e ^ -15